Escuela
Politécnica
Superior

UAPlant: Aplicación móvil para la realización de itinerarios botánicos

Máster Universitario en Desarrollo de Software para Dispositivos Móviles

Trabajo Fin de Máster

Autor:
Pablo López Iborra
Tutor/es:
Miguel Ángel Lozano Ortega
María Ángeles Alonso Vargas

Octubre 2020

Universitat d'Alacant
Universidad de Alicante

UAPlant: Aplicación móvil para la realización de itinerarios botánicos

Desarrollo de las aplicaciones nativas iOS y Android para la realización de itinerarios botánicos en el Campus de la Universidad de Alicante

Autor
Pablo López Iborra

Tutor/es
Miguel Ángel Lozano Ortega
Departamento de Ciencia de la Computación e Inteligencia Artificial
María Ángeles Alonso Vargas
Departamento de Ciencias Ambientales y Recursos Naturales

Máster Universitario en Desarrollo de Software para Dispositivos Móviles

Escuela Politécnica Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Octubre 2020

Preámbulo

Este proyecto propone realizar la aplicación UAPlant tanto para iOS como para Android, donde los alumnos de la Universidad de Alicante (UA), de la asignatura de Botánica del Grado en Biología, podrán hacer uso de ella para la realización de actividades destinadas al aprendizaje mediante preguntas. Con esta aplicación se pretende facilitar el estudio y aprendizaje de una parte importante de los conocimientos impartidos en la asignatura de Botánica, haciendo que el usuario aprenda interactuando directamente con el medio ambiente.

Agradecimientos

Ha sido un año largo, lleno de imprevistos, de situaciones nuevas, una pandemia mundial que ha azotado a todas las personas, pero aun con ello, con las dificultades encontradas durante todo el confinamiento, agradecer a todas las personas que han estado a mi lado, en especial a mis padres por todo lo que han hecho por mí durante todo este tiempo y por toda la confianza que han depositado en mí, a mi hermano por su apoyo, sus consejos y por toda la confianza y ganas que me ha transmitido durante todo este tiempo, a mis amigos por ser gran parte del apoyo y por toda la ayuda y confianza que han depositado en mí durante muchos de los días complicados, y por último agradecer en especial a mi pareja todo su apoyo, toda la confianza que ha depositado en mí, su paciencia durante horas y días, sus risas y su alegría durante días duros, ya que gracias a ella días muy difíciles han podido llegar a ser días muy buenos. También dar las gracias a mi tutor Miguel Ángel Lozano Ortega por su ayuda y paciencia durante todo el proyecto.

Agradecer a todas estas personas cercanas a mí por todo lo que han depositado en mi y por todo el amor que me han transmitido durante todo este tiempo, porque aunque haya sido un año difícil, un desarrollo largo y duro, gracias a todos ellos, todo esto ha llegado a ser posible, finalizando el desarrollo positivamente y de forma satisfactoria.

Índice general

Agradecimientos **vii**

1 Introducción **1**
1.1 Visión General . . . 1
1.2 Motivación . . . 1
1.3 Campos a tratar . . . 1
1.4 Objetivos . . . 2
1.5 Contenido . . . 2

2 Estado del arte **3**
2.1 Introducción . . . 3
2.2 Aplicaciones relacionadas . . . 3
2.3 Reconocimiento de imágenes . . . 5
2.4 Botánica . . . 8

3 Materiales y Métodos **11**
3.1 Introducción . . . 11
3.2 Planificación . . . 11
3.3 Materiales . . . 12
3.4 iOS . . . 13
3.4.1 Introducción . . . 13
3.4.2 Versiones . . . 14
3.4.2.1 iPhone OS X . . . 14
3.4.2.2 iOS X . . . 15
3.4.3 Arquitectura . . . 18
3.5 Android . . . 19
3.5.1 Introducción . . . 19
3.5.2 Versiones . . . 20
3.5.2.1 Android X . . . 21
3.5.3 Arquitectura . . . 23

4 Análisis y especificación **25**
4.1 Descripción general . . . 25
4.2 Funcionalidades de la aplicación . . . 28
4.3 Especificación de requisitos . . . 29
4.3.1 Requisitos funcionales . . . 29
4.3.1.1 Interfaz de usuario . . . 33
4.3.2 Requisitos no funcionales . . . 33

5 Desarrollo **35**
5.1 Introducción . . . 35
5.2 Servidor y fichero JSON . . . 35
5.3 iOS . . . 37
5.3.1 Visión general . . . 37
5.3.2 Creación del proyecto . . . 39
5.3.3 Pantalla inicial . . . 40
5.3.4 Pantalla de lista de itinerarios . . . 41
5.3.5 Pantalla de lista de plantas . . . 42
5.3.6 Mapa con actividades . . . 43
5.3.7 Ventana con significado del marcador . . . 45
5.3.8 Ventana informativa con significado de los marcadores . . . 45
5.3.9 Remodelación del modelo de datos . . . 46
5.3.10 Pantalla con la información de la actividad . . . 47
5.3.11 Pantalla con la lista de preguntas y respuestas del test . . . 49
5.3.12 Ventanas de información sobre el resultado del test . . . 49
5.3.13 Pantalla con la información de la planta . . . 50
5.3.14 Pantalla de Créditos y cambio de diseño e iconos de la aplicación . . . 52
5.3.15 Conexión y lectura de datos del servidor . . . 53
5.3.16 Gestor de descarga . . . 54
5.3.17 Pantalla y banner de descarga de archivos . . . 54
5.3.18 Ventana de información del borrado de un itinerario . . . 55
5.3.19 Envío de una notificación de error mediante correo electrónico . . . 56
5.4 Android . . . 59
5.4.1 Visión general . . . 59
5.4.2 Creación del proyecto . . . 59
5.4.3 Pantalla inicial . . . 60
5.4.4 Pantalla de lista de itinerarios . . . 61
5.4.5 Pantalla de lista de plantas . . . 64
5.4.6 Mapa con actividades . . . 65
5.4.7 Ventana con significado del marcador . . . 66
5.4.8 Ventana informativa con significado de los marcadores . . . 66
5.4.9 Pantalla con la información de la actividad . . . 67
5.4.10 Pantalla con la lista de preguntas y respuestas del test . . . 69
5.4.11 Ventanas de información sobre el resultado del test . . . 69
5.4.12 Pantalla con la información de la planta . . . 70
5.4.13 Pantalla de Créditos . . . 70
5.4.14 Diseño e iconos de la aplicación . . . 72
5.4.15 Conexión y lectura de datos del servidor . . . 72
5.4.16 Gestor de descarga . . . 73
5.4.17 Pantalla y banner de descarga de archivos . . . 74
5.4.18 Borrado de un itinerario . . . 75
5.4.19 Envío de una notificación de error mediante correo electrónico . . . 76

6 Resultados **77**

7 Conclusiones **83**

Bibliografía **85**

Lista de Acrónimos y Abreviaturas **87**

Índice de figuras

2.1 Imagen promocional de la app Google Lens, obtenida de Google Play. 4
2.2 Imagen de Google Play de la aplicación Pl@ntNet. 4
2.3 Imagen de la aplicación Arbolapp, obtenida de Google Play. 4
2.4 Imagen de Google Play de la app Picture This. 4
2.5 Esquema de las diferentes arquitecturas VGG. La columna "E" corresponde a la red con arquitectura VGG-19. Imagen obtenida de *www.quora.com*. 5
2.6 Ejemplo de itinerario botánico "Gimnospermas". Imagen realizada por la cotutora y profesora titular del Departamento de Ciencias Ambientales y Recursos Naturales (DCARN), María Ángeles Alonso Vargas. 9
2.7 Ejemplo de esquema sobre claves dicotómicas. Imagen de realización propia. . 9

3.1 Diagrama de Gantt explicativo sobre el periodo de desarrollo del Trabajo Final de Máster (TFM) 11
3.2 Arquitectura de capas de iOS. Imagen obtenida de *Arquitectura iOS* (s.f.) . . 18
3.3 Arquitectura de capas de Android. Imagen obtenida de Wikipedia Android. . 23

4.1 Mockup inicial de la aplicación. 26
4.2 Estructura principal planteada inicialmente. 27
4.3 Planteamiento inicial del modelo de datos. 28
4.4 Paleta de colores utilizados en la aplicación. 33

5.1 short 38
5.2 Creación del proyecto iOS. 39
5.3 Primera instancia de la pantalla inicial. 40
5.4 Celda final de la lista de itinerarios. 41
5.5 Celda final de la lista de plantas. 43
5.6 Google Maps como mapa principal para mostrar las actividades. 43
5.7 Ventana de información del marcador seleccionado. 45
5.8 Botón de información de los "markers" en la barra de navegación. 45
5.9 Pantalla informativa de los "markers" y los estados de las actividades. 46
5.10 Base de datos resultante de una nueva iteración sobre la figura 4.3. 46
5.11 Pestaña de información sobre la actividad pulsada. 47
5.12 Diagrama de flujo de cambio de estado de la actividad. 48
5.13 Celdas disponibles en el controlador de las preguntas del test. 49
5.14 DropDown para las respuestas del test. 49
5.15 Alerta indicativa de que el usuario ha fallado alguna respuesta. 50
5.16 Alerta indicativa de que el usuario ha finalizado el test correctamente. 50
5.17 Pantalla de información sobre una planta. 51

5.18 Alerta indicativa de que la planta seleccionada esta bloqueada y que actividad debe completar para desbloquearla. 51
5.19 Tab Bar final de la pagina principal. 52
5.20 Diseño de las secciones de la lista de itinerarios. 53
5.21 Diseño de las secciones de la lista de plantas. 53
5.22 Pantalla de descarga de archivos en la pantalla de bienvenida de la aplicación. 55
5.23 Banner de descarga de archivos en la lista de itinerarios. 55
5.24 Barra de navegación de la lista de itinerarios con el botón para realizar las descargas del servidor. 55
5.25 Alerta de borrado de un itinerario 56
5.26 Barra de navegación final con el botón de reporte. 58
5.27 Creación del proyecto Android. 59
5.28 Pantalla inicial de la aplicación en Android. 60
5.29 Barra de navegación con las tres pestañas principales. 61
5.30 Pantalla de la lista de itinerarios. 65
5.31 Pantalla de la lista de plantas. 65
5.32 Mapa de Google Maps con una actividad. 66
5.33 Ventana de información del marcador seleccionado. 66
5.34 Botón de información de los marcadores en la barra de navegación. 67
5.35 Pantalla informativa de los marcadores y los estados de las actividades. . . . 67
5.36 Pestaña de información sobre la actividad pulsada. 68
5.37 Diagrama de flujo de cambio de estado de la actividad. 68
5.38 Diseño preguntas, mostrando una preguntas y el botón de enviar al final de la lista. 69
5.39 Alerta indicativa de que el usuario ha fallado alguna respuesta. 69
5.40 Alerta indicativa de que el usuario ha finalizado el test correctamente. 70
5.41 Pantalla de información sobre una planta. 71
5.42 Alerta indicativa de que la planta seleccionada esta bloqueada y que actividad debe completar para desbloquearla. 71
5.43 Pantalla de créditos con información relevante sobre el desarrollo de la aplicación. 71
5.44 Diseño de las secciones de la lista de itinerarios. 72
5.45 Diseño de las secciones de la lista de plantas. 72
5.46 Pantalla de descarga de archivos en la pantalla de bienvenida de la aplicación. 74
5.47 Banner de descarga de archivos en la lista de itinerarios. 74
5.48 Barra de navegación de la lista de itinerarios con el botón para realizar las descargas del servidor. 75
5.49 Alerta de borrado de un itinerario 75
5.50 Barra de navegación final con el botón de reporte. 76

6.1 Pantalla inicial. 77
6.2 Lista de itinerarios con un itinerario. 78
6.3 Lista de plantas con una planta bloqueada. 78
6.4 Mapa con una actividad y su información. 79
6.5 Pantalla con los detalles de la actividad elegida. 79
6.6 Test con la lista de preguntas de la actividad. 80

6.7 Ventana informativa de que se ha fallado alguna respuesta en el test. 80
6.8 Ventana informativa de que se ha completado el test correctamente. 81
6.9 Lista de plantas con una planta desbloqueada. 81
6.10 Pantalla con la información de la planta seleccionada. 82
6.11 Pantalla de créditos con información sobre el desarrollo de la aplicación. . . . 82

Índice de tablas

3.1 Recopilatorio de versiones lanzadas de iPhone OS e iOS. 14
3.2 Versiones lanzadas de iPhone OS desde el año 2007 hasta el año 2010. 15
3.3 Versiones lanzadas de iOS desde el año 2010 hasta el año 2014. 15
3.4 Versiones lanzadas de iOS desde el año 2014 hasta el año 2018 (excepto algunas últimas actualizaciones en 2019). 16
3.5 Versiones estables lanzadas de iOS desde el año 2018 hasta el año 2020. . . . 17
3.6 Recopilatorio de versiones lanzadas de Android. 20
3.7 Versiones lanzadas de Android desde el año 2008 hasta el año 2012. 21
3.8 Versiones lanzadas de Android desde el año 2011/2012 hasta el año 2020. . . 22

Índice de Códigos

2.1 Extración de características 6
2.2 Entrenamiento de un clasificador 7

5.1 Ejemplo de archivo index.txt del servidor 35
5.2 Ejemplo de archivo gimnospermas.json 36
5.3 Arrays de itinerarios 41
5.4 Gestión de la lista de las plantas guardadas en Core Data 42
5.5 Gestión de la lista de las plantas guardadas en Core Data 44
5.6 Ejemplo de petición de datos al servidor mediante URLSession 53
5.7 Ejemplo de guardado de los datos de la entidad "Route" 53
5.8 Diccionarios donde se almacenan las urls de las imágenes a descargar en el gestor 54
5.9 Guardado del contexto de la aplicación desde una función sin acceso al contexto abierto previamente 54
5.10 Búsqueda de un itinerario en Core Data para su posterior eliminación 56
5.11 Añadido botón de reporte a la barra de navegación 56
5.12 Funciones para comprobar el gestor de correo electrónico a utilizar y para enviar el correo 57
5.13 Datos almacenados en la clase RouteListItem para mostrar en el CardView de la lista de itinerarios 62
5.14 Datos almacenados en la clase RoutesSection para mostrar en la sección de la lista de itinerarios 62
5.15 Clase RouteChildAdapterList para la gestión los ítems de las secciones de la lista de itinerarios 62
5.16 Clase RouteAdapterList para la gestión de las secciones de la lista de itinerarios 64
5.17 LinkedHashMap para almacenar los marcadores y las actividades correspondientes del mapa 66
5.18 Ejemplo de petición de datos al servidor mediante HttpURLConnection . . . 72
5.19 Ejemplo de guardado de los datos de la entidad "Route" 73
5.20 Diccionarios LinkedHashMap donde se almacenan las urls de las imágenes a descargar en el gestor 73
5.21 OnLongClickListener para el borrado de un itinerario en onBindViewHolder del RouteChildAdapterList 75
5.22 Gestor de reporte por correo electrónico 76

1 Introducción

1.1 Visión General

En este TFM nos centraremos en la creación de la aplicación móvil UAPlant en colaboración con Departamento de Ciencias Ambientales y Recursos Naturales (DCARN) de la Universidad de Alicante (UA). Inicialmente a esta propuesta se le sumaba un apartado de Inteligencia Artificial (IA) con el que el usuario mediante una fotografía realizada a la planta o árbol deseado podría averiguar de cuál se trataba, ya que se realizaría un reconocimiento de imagen mediante una red neuronal, dándole al usuario de esta manera un método eficaz, divertido y didáctico con el que aprender sobre botánica, en especial sobre las plantas y especies naturales del campus de la UA, pero debido a la situación por el Covid-19, se decidió cambiar este método de aprendizaje por el uso de preguntas y respuestas sobre *Claves Dicotómicas*, además de no solo realizar esta aplicación para iOS, sino también añadiéndole la versión de Android. Para la realización de sendas aplicaciones, utilizaremos los entornos de desarrollo nativos como son Xcode para iOS y Android Studio para Android. De esta manera se ha podido adaptar una idea de aplicación a ambos sistemas operativos, pudiendo de esta manera aprovechar los servicios en especifico de cada uno de ellos.

1.2 Motivación

La elección de este trabajo ha estado muy condicionada inicialmente por mi gusto personal por la IA y por el desarrollo de software orientado a poder ayudar a gente o facilitar el uso de métodos más tradicionales mediante nuevas tecnologías. Dado que este proyecto finalmente no ha hecho uso del apartado de IA por la imposibilidad de realizar pruebas debido al Covid-19, me he centrado especialmente en el desarrollo de una aplicación útil, eficaz e innovadora para el aprendizaje, de en este caso, sobre Botánica.

1.3 Campos a tratar

A cada año que pasa las personas somos cada vez más dependientes de la tecnología, lo cual la hace más importante en nuestras vidas, por lo que de esta manera nos hemos centrado en crear UAPlant como sustitutivo de métodos más tradicionales como son el lápiz y el papel. Para el desarrollo de este software, se hace uso del lenguaje de programación *Swift* para el sistema operativo de iOS, y *Java* en el caso de Android.

Dado que la aplicación es sobre el campo de la Biología, más concretamente el de la Botánica, se ha llevado a cabo una pequeña investigación en busca de diferentes aplicaciones relacionadas con estos campos, de donde se ha observado su forma de presentación, colores y otros factores sobre los que se han decidido a la hora de crear la aplicación lo más precisa y acorde al tema a tratar.

Más concretamente los campos sobre los que se trabaja en este proyecto son:

- Desarrollo de software para dispositivos móviles
  - iOS
  - Android
- Biología
  - Botánica

1.4 Objetivos

El objetivo principal de UAPlant es mejorar y facilitar el aprendizaje de los usuarios en el campo de la Botánica. Para ello se han marcado diversos objetivos necesarios:

- Búsqueda de aplicaciones relacionadas en el campo de la Botánica
- Planteamiento de posibles formas de aprendizaje basadas en las nuevas tecnologías
- Pruebas y entrenamiento con redes neuronales para el reconocimiento de imágenes de plantas [1]
- Entendimiento de la clasificación de las plantas mediante claves dicotómicas [2]
- Desarrollo de la aplicación final
  - iOS
  - Android[2]
- Adicción de la red neuronal entrenada a la aplicación final [1]

1.5 Contenido

En este documento seguiremos la siguiente estructura: En el Capítulo 1 se introduce el proyecto que se ha llevado a cabo. En el Capítulo 2 se puede observar el estudio que se ha realizado sobre los campos a tratar en este proyecto. En el Capítulo 3 se realiza una explicación de los materiales que se usan a lo largo del proyecto, además del hardware específico del que se hará uso, así como una explicación sobre los diferentes sistemas operativos que se van a utilizar. El Capítulo 4 explica como fue la fase de "Planteamiento de la app", detallando además las funcionalidades y los requisitos de la aplicación. En el Capítulo 5 se puede observar como se ha realizado el desarrollo para ambos sistemas operativos iOS y Android. El Capítulo 6 muestra capturas del resultado de ambas aplicaciones. Por último, en el Capítulo 7 se pueden leer las conclusiones finales del trabajo.

[1] Pospuesto debido a la situación actual con el Covid-19.
[2] Añadido debido a la eliminación del reconocimiento de imagen mediante red neuronal.

2 Estado del arte

2.1 Introducción

En esta sección se va a proceder a explicar la información buscada sobre elementos relacionados con los temas a tratar. Una vez se ha comenzado la fase *"Planteamiento de la app"*, en la cual se especifican y detallan las funciones, el diseño y todo lo relacionado con la app, se procede a la búsqueda de información sobre:

- **Aplicaciones:** Software de botánica para dispositivos móviles.
- **Inteligencia Artificial:** Reconocimiento de imágenes mediante redes neuronales.
- **Botánica:** Claves dicotómicas, itinerarios botánicos, etc.

2.2 Aplicaciones relacionadas

Las apps que se van a nombrar posteriormente, han servido de inspiración y de guía para el planteamiento de UAPlant, ya que se ha tenido en cuanto como los desarrolladores han utilizado algunas funcionalidades que nosotros también queremos implementar, y el diseño empleado en estas mismas. Sobre las diferentes apps encontradas sobre el reconocimiento de plantas o sobre el aprendizaje en el área de la botánica, *Arquitectura iOS* (s.f.), podemos encontrar una gran diversidad, pero de forma destacable podemos comentar las funcionalidades de:

- **Google Lens** Aplicación desarrollada por Google, la cual permite realizar una fotografía a la planta o árbol deseado. Una vez ha procesado esta imagen, nos enviará de forma casi inmediata a internet, informándonos del nombre de la planta o árbol, además de mostrarnos imágenes relacionadas. Podemos observar la app en la Figura 2.1.
- **Pl@ntNet app** Esta aplicación, al igual que Google Lens, permite obtener el nombre de la planta o árbol mediante una fotografía, pero además de esto también nos dará información relevante y nos sugerirá el nombre de otras plantas o arboles con características similares. Nos permite añadir más de una fotografía para realizar el reconocimiento. Podemos observar una imagen de la app en la Figura 2.2.
- **Arbolapp** Esta aplicación permite identificar un árbol mediante sus hojas. Contiene una base de datos muy amplia donde almacena todo tipos de hojas, por lo que para identificar el árbol deseado únicamente se tendrá que ir seleccionando las hojas deseadas, y la app mostrará el árbol resultante. Podemos observar la app en la Figura 2.3.

- **Picture This** Al igual que el resto de aplicaciones ya nombradas, esta aplicación permite reconocer una planta o árbol mediante una fotografía, pero además identifica plantas en mal estado, permite crear recordatorios y contiene un chat para hablar con expertos. Imagen de la app en la Figura 2.4.

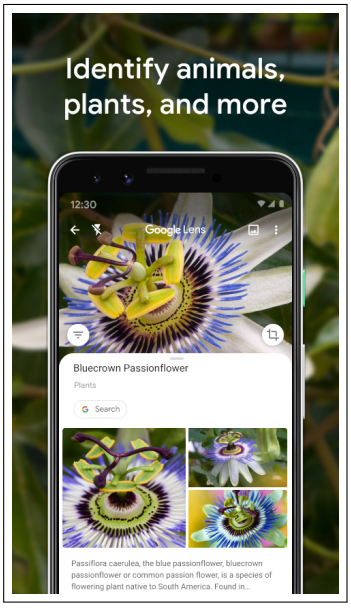

Figura 2.1: Imagen promocional de la app Google Lens, obtenida de Google Play.

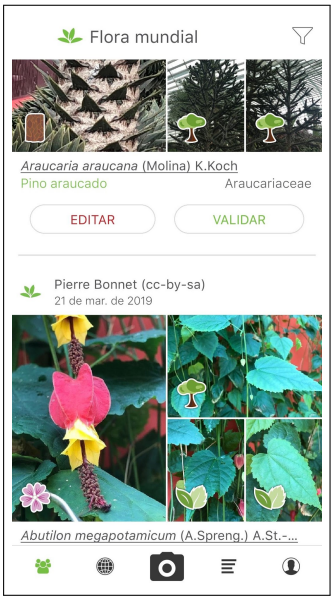

Figura 2.2: Imagen de Google Play de la aplicación Pl@ntNet.

Figura 2.3: Imagen de la aplicación Arbolapp, obtenida de Google Play.

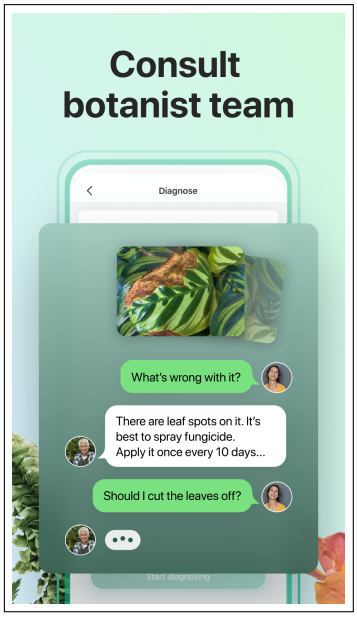

Figura 2.4: Imagen de Google Play de la app Picture This.

Visto que las principales aplicaciones que utilizan el reconocimiento de imagen como herramienta principal de identificación, nuestro objetivo era incluir ese mismo reconocimiento como una herramienta de diversión para el usuario, con el objetivo de que tuviera que buscar esa planta dentro de la ruta propuesta para poder validarla y desbloquearla.

2.3 Reconocimiento de imágenes

Para poder realizar el reconocimiento de plantas mediante imágenes, se han realizado pruebas mediante una red *VGG-19* preentrenada, con el posterior objetivo de recopilar imágenes para crear una base de datos con la que entrenar nuestra propia red.

- **VGG-19:** Se trata de una red neuronal convolucional preentrenada con 19 capas de profundidad, de las cuales 16 son capas convolucionales y las 3 restantes son capas totalmente conectadas. Con esta red podemos entrenar diferentes clasificadores.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 2.5: Esquema de las diferentes arquitecturas VGG. La columna "E" corresponde a la red con arquitectura VGG-19. Imagen obtenida de *www.quora.com*.

Para la ejecución de esta red neuronal se ha hecho uso de la api de *Keras*, la cual se trata de una biblioteca de redes neuronales de código abierto en Python. Debido a la situación actual por el Covid-19, únicamente se han podido realizar pruebas con una base de datos existente, y después de realizar diferentes pruebas con varios métodos para clasificación de imágenes, se determinó que los mejores resultados se obtenían con la red VGG-19 y con la librería *SKLearn*. Para realizar esta prueba se ha procedido a la simulación del ejemplo que podemos encontrar en el documento: Anubha Pearline y cols. (2019).

Código 2.1: Extración de características

```
from os import listdir
from os import path
from random import shuffle
from numpy import array
from pickle import load
from pickle import dump
from keras.applications.vgg19 import VGG19
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.applications.vgg19 import preprocess_input
from keras.models import Model
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint

def build_model():
    # Cargamos el modelo VGG19
    model = VGG19()
    # Eliminamos la última capa de la red
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    # Muestra resumen de la red
    return model


def feature_extraction(image_filename, model):
  # Extrae las caracteristicas de cada foto a partir de VGG19
  image = load_img(image_filename, target_size=(224, 224))
  image = img_to_array(image)
  image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
  image = preprocess_input(image)
  feature = model.predict(image, verbose=0)
  feature = feature.reshape(feature.shape[0] * feature.shape[1])
```

```
  return feature


def read_dataset(directory):
  features = dict()
  categories = set()
  model = build_model()

  for category in listdir(directory):
    categories.add(category)
    category_dir = directory + '/' + category
    for image_name in listdir(category_dir):
      image_file = category_dir + '/' + image_name
      f = feature_extraction(image_file, model)
      features[image_name] = { 'category': category, 'features': f }
  return features, categories


feats, categories = read_dataset('Folio Leaf Dataset/Folio/')
dump(feats, open('features.pkl', 'wb'))
```

Código 2.2: Entrenamiento de un clasificador

```
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from numpy.random import permutation
from numpy import arange

# Carga las caracteristicas y categorias
feat_dict = load(open('features.pkl', 'rb'))
data = [(feat_dict[image]['features'], feat_dict[image]['category']) for image in feat_dict]
Xt, yt = zip(*data)
X = array(Xt)
y = array(yt)

# Separa conjunto de test y de train
total_samples = len(X)
train_samples = int(0.7 * total_samples)
shuffle_indices = permutation(arange(total_samples))
train_indices = shuffle_indices[:train_samples]
test_indices = shuffle_indices[train_samples:]

# Entrena y evalúa el clasificado
clf = LogisticRegression(random_state=0, max_iter=1000).fit(X[train_indices, :], y[train_indices])
y_pred = clf.predict(X[test_indices, :])
print('Test score', accuracy_score(y[test_indices], y_pred))
```

Después de ejecutar esta simulación en Google Colab, observamos que tiene un índice de acierto de 0.95 sobre 1 debido al clasificador *Logistic Regression (LD)*, por lo que podemos afirmar que este método de clasificación es muy efectivo con las diferentes plantas de la base de datos, de las cuales extraemos un vector de 4096 características de cada imagen con el objetivo de entrenar este clasificador *LD*.

2.4 Botánica

Dada la situación actual, el no poder agregar a la aplicación la herramienta de reconocimiento de imágenes supone un gran cambio para el flujo de la aplicación, por lo que tras diversas reuniones se procuró encontrar algo que permitirá a los usuarios poder hacer uso de la aplicación, y tras varias propuestas se llegó a la conclusión de que lo más eficiente podía ser realizar un test sobre la planta a desbloquear, y así evitar principalmente que alumnos de Biología tuvieran que continuar realizando un cuaderno con las clasificaciones individuales de cada planta a papel. Para lograr este objetivo se acordó realizar las preguntas del test basándonos en las *Claves Dicotómicas* de cada planta. Como este proyecto está en colaboración con Departamento de Ciencias Ambientales y Recursos Naturales (DCARN), se ha podido obtener información en especifico sobre claves dicotómicas, *Clave dicotómica* (s.f.), de las plantas de Universidad de Alicante (UA), lo cual nos ha permitido también entender un poco mejor el funcionamiento de estas para poder realizar y organizar el test de la mejor manera posible. Para también entender de mejor manera el trabajo que realizan los alumnos de botánica con el cuaderno de clasificaciones mediante los diferentes *itinerarios botánicos* que se les proponen, se va a realizar en primer lugar un trabajo de búsqueda y entendimiento de los términos necesarios.

- **Itinerarios Botánicos:** Conjunto de rutas específicas, en un orden determinado, que sirven al alumno para el aprendizaje de las especies que se encuentran en el campus de la UA. Sobre estos itinerarios deben realizar un cuaderno de aprendizaje, donde deben apuntar todo lo relacionado sobre las plantas de estas rutas. Podemos observar un ejemplo en la figura 2.6.
- **Claves Dicotómicas:** Herramienta que permite distinguir organismos mediante *categorías taxonómicas* especificas de cada uno. Aquí podemos distinguir algunas de estas categorías como son: especie, género, familia, etc. Estas claves se basan en definiciones de los caracteres morfológicos, macroscópicos o microscópicos de estos organismos. Ejemplo de esquema en la figura 2.7

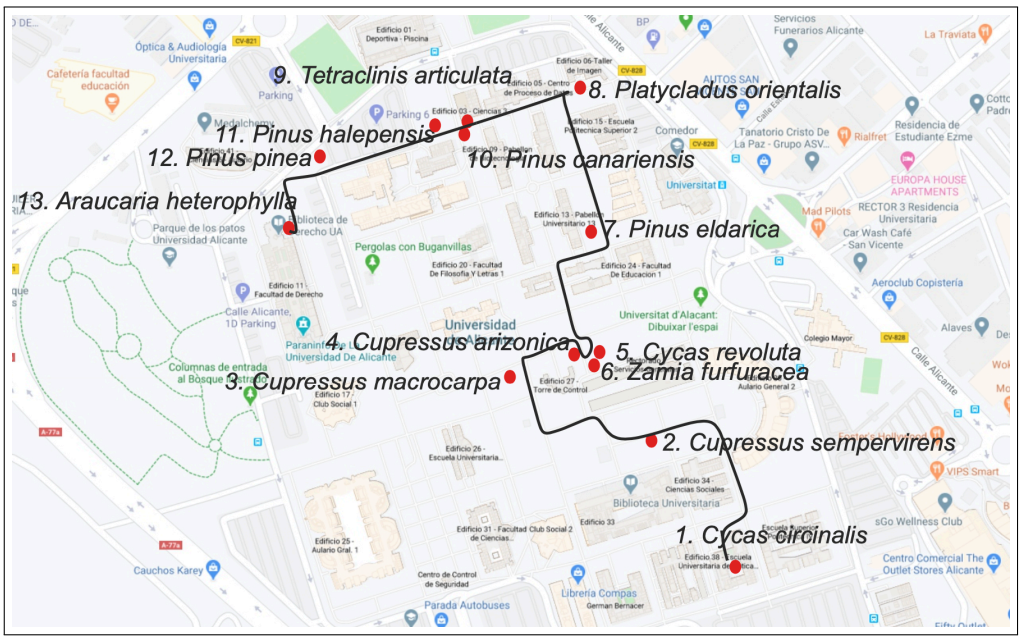

Figura 2.6: Ejemplo de itinerario botánico "Gimnospermas". Imagen realizada por la cotutora y profesora titular del DCARN, María Ángeles Alonso Vargas.

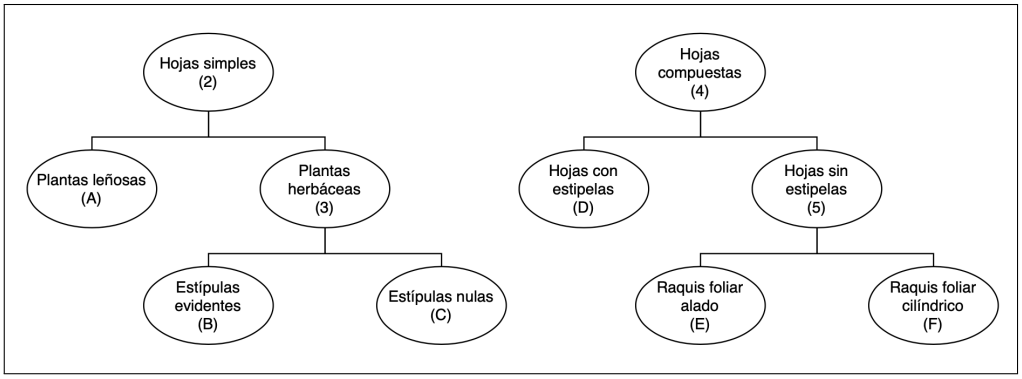

Figura 2.7: Ejemplo de esquema sobre claves dicotómicas. Imagen de realización propia.

3 Materiales y Métodos

3.1 Introducción

En este capítulo hablaremos de los diferentes sistemas operativos sobre los que vamos a trabajar para desarrollar **UAPlant**. En primer lugar se citarán y explicarán los materiales usados para llevar a cabo este proyecto, y posteriormente se realizará una introducción a cada sistema operativo, para después explicar con más detalle los requerimientos de cada uno de ellos para funcionar correctamente.

Los sistemas operativos a tratar son:

- **iOS:** Sistema operativo de *Apple Inc*, el cual fue desarrollado en primera instancia para iPhone, y utilizado posteriormente en cada uno de sus dispositivos portables. Este sistema operativo no puede ser instalado en hardware de terceros.
- **Android:** Desarrollado inicialmente por *Android Inc*, y posteriormente comprado por *Google*. Este sistema operativo es de código abierto y fue creado para su utilización en dispositivos portátiles con pantalla táctil.

3.2 Planificación

En este apartado se va a explicar el proceso que se ha llevado a cabo para el desarrollo completo del proyecto de trabajo del TFM. Para dar apoyo a la explicación que se va a realizar, se va a utilizar un diagrama de Gantt donde se expondrán todos los plazos que se han tomado de forma aproximada, esto podemos verlo en la Figura 3.1.

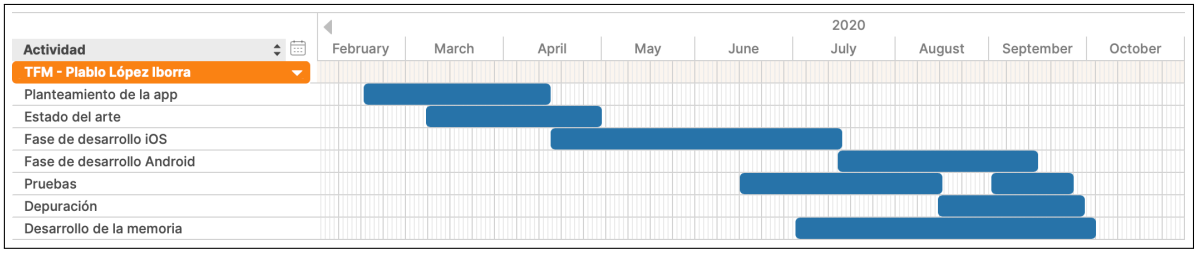

Figura 3.1: Diagrama de Gantt explicativo sobre el periodo de desarrollo del TFM

Como se ha podido observar en el diagrama anterior, el proceso de trabajo consta de varias etapas:

- **Planteamiento de la aplicación:** En primer lugar se realizaron diversas reuniones o meetings para direccionar la propuesta de la app conforme a las necesidades del DCARN, encargados de la asignatura de Botánica en la UA.

- **Estado del arte:** En segundo lugar se procedió a realizar una breve investigación sobre aplicaciones relacionadas con la Botánica, buscando de esta manera relaciones entre ellas que nos pudieran servir a la hora de diseñar nuestra app, tales como la presentación, los colores y otros factores diversos de estas mismas.
- **Fase de desarrollo iOS:** En esta fase se ha procedido a desarrollar por completo la app para iOS, de manera que se ajustara a la propuesta planteada en la primera fase *Planteamiento de la aplicación*.
- **Fase de desarrollo Android:** Posteriormente a la realización de la app para iOS, se ha procedido a realizar la app para el sistema operativo Android, la cual sigue el planteamiento y la estructura utilizada en la app de iOS, de manera que se tuviera un conjunto de apps con la mayor sinergia posible.
- **Pruebas:** Esta fase tiene como duración una gran parte del periodo de desarrollo de ambas apps, ya que durante el proceso de diseño y creación de estas, se ha debido ir probando en todo momento la funcionalidad y la adaptación a los diferentes tipos de dispositivos. Además se ha añadir que durante este proceso también se ha tenido que ir probando la conectividad con el servidor web que almacena todos los datos que posteriormente serán recogidos por ambas aplicaciones.
- **Depuración:** Fase de breve duración donde se realizan pruebas finales únicamente para comprobar la consistencia y funcionalidad completa de ambas aplicaciones, ya que las pruebas más exhaustivas han sido realizadas en la fase de *Pruebas*.
- **Desarrollo de la memoria:** Periodo estimado para la realización de la memoria del proyecto, la cual se redacta al mismo tiempo que se realiza el desarrollo y las diversas pruebas.

3.3 Materiales

Para llevar a cabo el desarrollo de UAPlant han sido necesarios los siguientes elementos:

- **Hardware**
  - **Ordenador:** Para llevar a cabo el desarrollo de UAPlant, se ha utilizado un *Apple MacBook Pro 15" i7 2,6GHz 2019*
  - **Dispositivos móviles:** Para la realización de pruebas se han utilizado los siguientes dispositivos físicos:
    - **iOS**
      - **Apple iPhone SE 2016** iOS v13.2.3
    - **Android**
      - **Xiaomi MI9 2019** Android v10 MIUI v12.0.1.0

- **Software:** Utilizaremos varios tipos de software para realizar diferentes tareas:
  - **Entornos de desarrollo**
    - **Xcode:** Entorno de desarrollo oficial con múltiples herramientas destinadas a la creación de software para dispositivos Apple.
    - **Android Studio:** Entorno de desarrollo integrado oficial, el cual dispone de múltiples herramientas para la creación de software para dispositivos Android.
  - **Almacenamiento en la nube**
    - **Github:** Sistema web que permite gestionar proyectos y llevar a cabo un control de las versiones de los archivos.
    - **Google Drive:** Servicio de alojamiento de archivos que permite al usuario acceder a ellos mediante un servicio web.

3.4 iOS

3.4.1 Introducción

Este sistema operativo fue presentado por Apple en la *Macworld Conference and Expo del 9 de enero de 2007* con el nombre de iPhone OS *"iPhone Operative System"*, lanzado en junio de ese mismo año con la primera versión del iPhone, aunque no tuvo nombre oficial hasta la salida del iPhone SDK el 6 de marzo de 2008, junto con la segunda versión del sistema operativo, la cual contendría por primera vez la App Store instalada. Este nombre se debió a que inicialmente este sistema operativo únicamente iba a ser lanzado para el iPhone, y posteriormente para el iPod Touch, pero el 4 de enero de 2010 Steve Jobs, CEO de Apple, anuncia que también va a lanzar el iPad, el cual estaría orientado a la industria de contenidos, con una mayor pantalla táctil, y con el mismo sistema operativo que su hermano mayor, por lo que el 7 de junio de 2010, durante la presentación del iPhone 4, se anuncia que iPhone OS pasaría a llamarse iOS, generalizando así el nombre del sistema operativo, el cual sería utilizado para los diferentes dispositivos portátiles de Apple, y el 21 de junio de este mismo año se lanzaría la primera versión oficial de iOS 4, la cual sería la primera actualización que ya no estaría disponible para las primeras generaciones de iPhone e iPod Touch.

A mediados de 2010 se estima que la App Store de iPhone OS tenía disponibles 185.000 aplicaciones funcionales.

A partir de ese momento Apple actualizaría año tras año su versión conocida de iOS, dejándonos a día de hoy la versión estable 13.7 y la versión 14 beta 8. Para llegar hasta este punto, Apple fue mejorando en cada actualización las funcionalidades de este sistema operativo, permitiendo al usuario cada vez una mayor interacción con el sistema.

iPhone OS X	
Versión	Fecha de lanzamiento
iPhone OS 1.0	29 de junio de 2007
iPhone OS 2.0	11 de julio de 2008
iPhone OS 3.0	17 de junio de 2009

iOS X	
Versión	Fecha de lanzamiento
iOS 4.x	21 de junio de 2010
iOS 5.x	12 de octubre de 2011
iOS 6.x	19 de septiembre de 2012
iOS 7.x	18 de septiembre de 2013
iOS 8.x	14 de septiembre de 2014
iOS 9.x	16 de septiembre de 2015
iOS 10.x	13 de septiembre de 2016
iOS 11.x	19 de septiembre de 2017
iOS 12.x	17 de septiembre de 2018
iOS 13.x	19 de septiembre de 2019

Tabla 3.1: Recopilatorio de versiones lanzadas de iPhone OS e iOS.

3.4.2 Versiones

Las versiones del principal sistema operativo de Apple pueden ser diferenciadas en dos categorías:

- **iPhone OS** → 2007-2010
- **iOS** → 2010-actualidad (septiembre 2020)

3.4.2.1 iPhone OS X

iPhone OS 1.0	
Versión	Fecha de lanzamiento
1.0	29 de junio de 2007
1.0.1	30 de julio de 2007
1.0.2	21 de agosto de 2007
1.1	14 de septiembre de 2007
1.1.1	27 de septiembre de 2007
1.1.4	26 de febrero de 2008
1.1.5	15 de julio de 2008

iPhone OS 2.0	
Versión	Fecha de lanzamiento
2.0	11 de julio de 2008
2.0.1	4 de agosto de 2008
2.0.2	18 de agosto de 2008
2.1	9 de septiembre de 2008
2.2	21 de noviembre de 2008
2.2.1	27 de enero de 2009

iPhone OS 3.0	
Versión	Fecha de lanzamiento
3.0	17 de junio de 2009
3.0.1	31 de julio de 2009
3.1	9 de septiembre de 2009
3.1.2	8 de octubre de 2009
3.1.3	2 de febrero de 2010
3.2	3 de abril de 2010
3.2.1	15 de julio de 2010
3.2.2	11 de agosto de 2010

Tabla 3.2: Versiones lanzadas de iPhone OS desde el año 2007 hasta el año 2010.

3.4.2.2 iOS X

iOS 4.x	
Versión	Fecha de lanzamiento
4.0	21 de junio de 2010
4.0.2	11 de agosto de 2010
4.1	8 de septiembre de 2010
4.2	22 de noviembre de 2010
4.3	9 de marzo de 2011

iOS 5.x	
Versión	Fecha de lanzamiento
5.0	12 de octubre de 2011
5.0.1	10 de noviembre de 2011
5.1	7 de marzo de 2012
5.1.1	7 de mayo de 2012

iOS 6.x	
Versión	Fecha de lanzamiento
6.0	19 de septiembre de 2012
6.0.1	1 de noviembre de 2012
6.0.2	18 de diciembre de 2012
6.1	28 de enero de 2013
6.1.1	6 de febrero de 2013
6.1.2	19 de febrero de 2013
6.1.3	19 de marzo de 2013
6.1.4	2 de mayo de 2013
6.1.5	14 de noviembre de 2013
6.1.6	23 de febrero de 2014

iOS 7.x	
Versión	Fecha de lanzamiento
7.0	18 de septiembre de 2013
7.0.1	19 de septiembre de 2013
7.0.2	26 de septiembre de 2013
7.0.3	22 de octubre de 2013
7.0.4	14 de noviembre de 2013
7.0.5	29 de enero de 2014
7.0.6	21 de febrero de 2014
7.1	10 de marzo de 2014
7.1.1	22 de abril de 2014
7.1.2	29 de junio de 2014

Tabla 3.3: Versiones lanzadas de iOS desde el año 2010 hasta el año 2014.

iOS 8.x	
Versión	Fecha de lanzamiento
8.0	14 de septiembre de 2014
8.0.1	24 de septiembre de 2014
8.0.2	25 de septiembre de 2014
8.1	20 de octubre de 2014
8.1.1	17 de noviembre de 2014
8.1.2	9 de diciembre de 2014
8.1.3	14 de enero de 2015
8.2	9 de marzo de 2015
8.3	8 de abril de 2015
8.4	30 de junio de 2015
8.5	13 de agosto de 2015

iOS 9.x	
Versión	Fecha de lanzamiento
9.0	16 de septiembre de 2015
9.0.1	23 de septiembre de 2015
9.0.2	30 de septiembre de 2015
9.1	21 de octubre de 2015
9.2	8 de diciembre de 2015
9.2.1	19 de enero de 2016
9.3	21 de marzo de 2016
9.3.1	31 de marzo de 2016
9.3.2	16 de mayo de 2016
9.3.3	18 de julio de 2016
9.3.4	4 de agosto de 2016
9.3.5	25 de agosto de 2016
9.3.6	22 de julio de 2019

iOS 10.x	
Versión	Fecha de lanzamiento
10.0	13 de septiembre de 2016
10.0.2	23 de septiembre de 2016
10.0.3	17 de octubre de 2016
10.1	24 de octubre de 2016
10.1.1	31 de octubre de 2016
10.2	12 de diciembre de 2016
10.2.1	17 de enero de 2017
10.3	27 de marzo de 2017
10.3.1	3 de abril de 2017
10.3.2	15 de mayo de 2017
10.3.3	19 de julio de 2017
10.3.4	22 de julio de 2019

iOS 11.x	
Versión	Fecha de lanzamiento
11.0	19 de septiembre de 2017
11.0.1	26 de septiembre de 2017
11.0.2	3 de octubre de 2017
11.0.3	11 de octubre de 2017
11.1	31 de octubre de 2017
11.1.1	9 de noviembre de 2017
11.1.2	16 de noviembre de 2017
11.2	2 de diciembre de 2017
11.2.1	13 de diciembre de 2017
11.2.2	8 de enero de 2018
11.2.5	23 de enero de 2018
11.2.6	19 de febrero de 2018
11.3	29 de marzo de 2018
11.3.1	24 de abril de 2018
11.4	29 de mayo de 2018
11.4.1	9 de julio de 2018

Tabla 3.4: Versiones lanzadas de iOS desde el año 2014 hasta el año 2018 (excepto algunas últimas actualizaciones en 2019).

iOS 12.x	
Versión	Fecha de lanzamiento
12.0	17 de septiembre de 2018
12.0.1	8 de octubre de 2018
12.0.2	7 de noviembre de 2018
12.1.1	5 de diciembre de 2018
12.1.2	20 de diciembre de 2018
12.1.3	22 de enero de 2019
12.1.4	7 de febrero de 2019
12.2	25 de marzo de 2019
12.3	13 de mayo de 2019
12.3.1	24 de mayo de 2019
12.3.2	10 de junio de 2019
12.4	22 de julio de 2019
12.4.1	21 de agosto de 2019
12.4.2	26 de septiembre de 2019
12.4.3	28 de octubre de 2019
12.4.4	10 de diciembre de 2019
12.4.5	28 de enero de 2020
12.4.6	24 de marzo de 2020
12.4.7	20 de mayo de 2020
12.4.8	15 de julio de 2020

iOS 13.x	
Versión	Fecha de lanzamiento
13.0	19 de septiembre de 2019
13.1	26 de septiembre de 2019
13.1.1	28 de septiembre de 2019
13.1.2	30 de septiembre de 2019
13.1.3	15 de octubre de 2019
13.2	28 de octubre de 2019
13.2.2	7 de noviembre de 2019
13.2.3	18 de noviembre de 2019
13.3	10 de diciembre de 2019
13.3.1	28 de enero de 2020
13.4	24 de marzo de 2020
13.4.1	7 de abril de 2020
13.5	20 de mayo de 2020
13.5.1	1 de junio de 2020
13.6	15 de julio de 2020
13.6.1	14 de agosto de 2020
13.7	1 de septiembre de 2020

Tabla 3.5: Versiones estables lanzadas de iOS desde el año 2018 hasta el año 2020.

Cabe destacar que en la Apple Worldwide Developers Conference (WWDC) de 2019 Apple anunció que los iPad pasarían a tener su propio sistema operativo iPadOS, debido a que el iPad se iba diferenciando cada vez más del resto de sus hermanos, por lo que iOS 13 sería lanzado junto con iPadOS 13.

3.4.3 Arquitectura

La arquitectura de iOS está formada por cuatro capas muy marcadas:

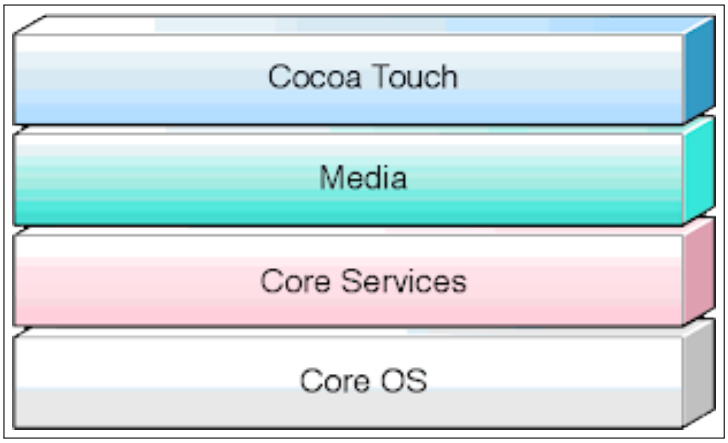

Figura 3.2: Arquitectura de capas de iOS. Imagen obtenida de *Arquitectura iOS* (s.f.)

- **Cocoa Touch:** Capa de alto nivel muy importante para los desarrolladores de aplicaciones, ya que contiene los dos principales frameworks: UIKit y Foundation Framework.
- **Media:** Capa de medio nivel que provee los servicios fundamentales de gráficos y multimedia a la capa superior *Cocoa Touch*.
- **Core Services:** Capa de medio-bajo nivel, la cual provee de los servicios fundamentales del sistema a las aplicaciones que lo necesiten.
- **Core OS:** Capa de bajo nivel donde se administran las características más básica del sistema: drivers, manejo de memoria, ficheros del sistema, seguridad y muchos más.

3.5 Android

3.5.1 Introducción

Este sistema operativo fue desarrollado por *Android Inc.*, pero fue Google la que respaldó económicamente la empresa, la cual adquirió en 2005 para hacerse cargo totalmente. Este sistema operativo fue desarrollado con el objetivo de ser un estándar entre los futuros teléfonos inteligentes que se proponían. Desarrollado con kernel basado en Linux v2.6, y de código abierto, diseñado para dispositivos móviles con pantalla táctil, Android Pie 1.0 fue presentado el 5 de noviembre de 2007 junto con la fundación *Open Handset Alliance*, creada previamente el mismo día de la presentación. Esta fundación se trata de un conglomerado de 78 fabricantes y desarrolladores de hardware, software y operadores de servicio, unidos para estandarizar un sistema operativo abierto con gran potencial futuro, que se declaraba como algo innovador en el mundo de los nuevos teléfonos inteligentes. El 23 de septiembre de 2008 se lanza la primera versión de Android Pie 1.0 junto con los nuevos teléfonos inteligentes, los cuales se ubicaron en el primer puesto en los Estados Unidos, en el segundo y tercer trimestres de 2010, con una cuota de mercado de 43.6% en el tercer trimestre. A escala mundial alcanzó una cuota de mercado del 50.9% durante el cuarto trimestre de 2011, más del doble que iOS de Apple, el segundo sistema operativo más globalizado para dispositivos móviles. Google liberó la mayor parte del código de Android bajo la licencia Apache, una licencia libre y de código abierto. Con el paso de los años Android ha ido ganando terreno en diferentes plataformas, soportando a día de hoy en 2020 dispositivos como teléfonos inteligentes, tabletas, relojes inteligentes (Wear OS), automóviles (Android Auto) y televisores (Android TV), con su arquitectura de hardware ARM hay soporte para sistemas x86 en el proyecto Android-x86, y Google TV utiliza una versión especial de Android x86. Como ya se ha comentado, lo más destacable dentro de Android es que se ha convertido en un sistema operativo estándar de código abierto, y esto también es posible gracias a que sus aplicaciones son ejecutadas en un framework Java, y su desarrollo está basado en diferentes tipos de lenguaje de programación abierto, formado por 12 millones de líneas de código, como podemos ver en Pearl Doughty-White (s.f.), y de librerías abiertas.

- **Lenguajes de programación**
  - **XML:** 3 millones de líneas de código.
  - **C:** 2.8 millones de líneas de código.
  - **Java:** 2.1 millones de líneas de código.
  - **C++:** 1.75 millones de líneas de código.
- **Librerias**
  - **Administrador de interfaz gráfica**
  - **Framework OpenCore**
  - **SQLite:** Base de datos relacional.
  - **OpenGL ES 2.0 3D:** Interfaz de programación de API gráfica.
  - **WebKit:** Motor de renderizado.

- **SSL:** Protocolos de transporte de datos.
- **SGL:** Motor gráfico.
- **Bionic:** Biblioteca estándar de C.

Uno de los elementos clave de Android es la máquina virtual **Dalvik**, que permite la traducción de Java bytecode, y la cual gestiona el hardware y los accesos a servicios o aplicaciones desde una capa de nivel intermedia, permitiendo a los desarrolladores tener una capa de abstracción que les asegura que nunca tendrán que preocuparse por una aplicación de hardware en particular. Esta máquina virtual es usada por Android hasta la versión 4.4.3, donde deja de usarse casi por completo para usar Android Runtime (ART), pero en la 5.0 de Android, Dalvik desaparece por completo dejando ART como único entorno de ejecución.

3.5.2 Versiones

Desde que se lanzó en 2008, Android no ha dejado de crecer cada vez más y más de la mano de Google, el cual le ha dado soporte en todo momento, haciendo de esa manera que mes a mes, año tras año, Android se siga actualizando con novedades y con nuevas funcionalidades útiles para el usuario, hasta llegar a ser lo que es hoy en día.

Android			
Nombre Código	Versión	Fecha de lanzamiento	Nivel de Api
Apple Pie	1.0	23 de septiembre de 2008	1
Banana Bread	1.1	9 de febrero de 2009	2
Cupcake	1.5	25 de abril de 2009	3
Donut	1.6	15 de septiembre de 2009	4
Eclair	2.0 - 2.1	26 de octubre de 2009	5 - 7
Froyo	2.2 - 2.2.3	20 de mayo de 2010	8
Gingerbread	2.3 - 2.3.7	6 de diciembre de 2010	9 - 10
Honeycomb	3.0 - 3.2.6	22 de febrero de 2011	11 - 13
Ice Cream Sandwich	4.0 - 4.0.5	18 de octubre de 2011	14 - 15
Jelly Bean	4.1 - 4.3.1	9 de julio de 2012	16 - 18
KitKat	4.4 - 4.4.4	31 de octubre de 2013	19 - 20
Lollipop	5.0 - 5.1.1	12 de noviembre de 2014	21 - 22
Marshmallow	6.0 - 6.0.1	5 de octubre de 2015	23
Nougat	7.0 - 7.1.2	15 de junio de 2016	24 - 25
Oreo	8.0 - 8.1	21 de agosto de 2017	26 - 27
Pie	9.0	6 de agosto de 2018	28
10	10.0	3 de septiembre de 2019	29
11	11.0	8 de septiembre de 2020	30

Tabla 3.6: Recopilatorio de versiones lanzadas de Android.

Como podemos observar, todas las versiones lanzadas, excepto las dos últimas hasta el momento en 2020, tienen como nombre en clave el nombre de un postre, además de ser lanzadas en orden alfabético. Como se ha comentado, Google decide el 3 de septiembre de 2019, durante la ceremonia en Mountain View, que las próximas actualizaciones dejarían de

tener nombres en clave a solo llamarse por su número de versión, de ahí la siguiente versión Android 10.

3.5.2.1 Android X

Android 1.0 Apple Pie	
Versión	Fecha de lanzamiento
1.0	23 de septiembre de 2008

Android 1.1 Banana Bread	
Versión	Fecha de lanzamiento
1.1	9 de febrero de 2009

Android 1.5 Cupcake	
Versión	Fecha de lanzamiento
1.5	27 de abril de 2009

Android 1.6 Donut	
Versión	Fecha de lanzamiento
1.6	15 de septiembre de 2009

Android 2.0/2.1 Eclair	
Versión	Fecha de lanzamiento
2.0	26 de octubre de 2009
2.0.1	3 de diciembre de 2009
2.1	12 de enero de 2010

Android 2.2 Froyo	
Versión	Fecha de lanzamiento
2.2	20 de mayo de 2010
2.2.1	18 de enero de 2011
2.2.2	22 de enero de 2011
2.2.3	21 de noviembre de 2011

Android 2.3 Gingerbread	
Versión	Fecha de lanzamiento
2.3.0/2.3.1	6 de diciembre de 2010
2.3.3	9 de febrero de 2011
2.3.4	28 de abril de 2011
2.3.5	25 de julio de 2011
2.3.6	2 de septiembre de 2011
2.3.7	21 de septiembre de 2011

Android 3.0 Honeycomb	
Versión	Fecha de lanzamiento
3.0	22 de febrero de 2011
3.1	10 de mayo de 2011
3.2	15 de julio de 2011
3.2.1	20 de septiembre de 2011
3.2.2	30 de septiembre de 2011
3.2.3	1 de diciembre de 2011
3.2.4	28 de febrero de 2012

Tabla 3.7: Versiones lanzadas de Android desde el año 2008 hasta el año 2012.

Android 4.0 Ice Cream Sandwich	
Versión	Fecha de lanzamiento
4.0.0/4.0.1	12 de octubre de 2011
4.0.2	29 de noviembre de 2011
4.0.3	16 de diciembre de 2011
4.0.4	8 de noviembre de 2012

Android 4.1 Jelly Bean	
Versión	Fecha de lanzamiento
4.1	9 de julio de 2012
4.1.1	23 de julio de 2012
4.1.2	9 de octubre de 2012

Android 4.2 Jelly Bean	
Versión	Fecha de lanzamiento
4.2	13 de noviembre de 2012
4.2.1	27 de noviembre de 2012
4.2.2	11 de abril de 2013

Android 4.3 Jelly Bean	
Versión	Fecha de lanzamiento
4.3	14 de julio de 2013

Android 4.4 KitKat	
Versión	Fecha de lanzamiento
4.4	31 de octubre de 2013
4.4.1	5 de diciembre de 2013
4.4.2	9 de diciembre de 2013
4.4.3	2 de junio de 2014
4.4.4	19 de junio de 2014

Android 5.0 Lollipop	
Versión	Fecha de lanzamiento
5.0	3 de noviembre de 2014
5.0.1	2 de diciembre de 2014
5.0.2	19 de diciembre de 2014
5.1	9 de marzo de 2015
5.1.1	19 de abril de 2015

Android 6.0 Marshmallow	
Versión	Fecha de lanzamiento
6.0	5 de octubre de 2015
6.0.1	7 de diciembre de 2015

Android 7.0 Nougat	
Versión	Fecha de lanzamiento
7.0	22 de agosto de 2016
7.1	20 de octubre de 2016

Android 8.0 Oreo	
Versión	Fecha de lanzamiento
8.0	21 de agosto de 2017
8.1	5 de diciembre de 2017

Android 9.0 Pie	
Versión	Fecha de lanzamiento
9.0	9 de agosto de 2018

Android 10.0	
Versión	Fecha de lanzamiento
10.0	3 de septiembre de 2019

Android 11.0	
Versión	Fecha de lanzamiento
11.0	8 de septiembre de 2020

Tabla 3.8: Versiones lanzadas de Android desde el año 2011/2012 hasta el año 2020.

3.5.3 Arquitectura

La arquitectura de Android está formada por cinco capas muy marcadas:

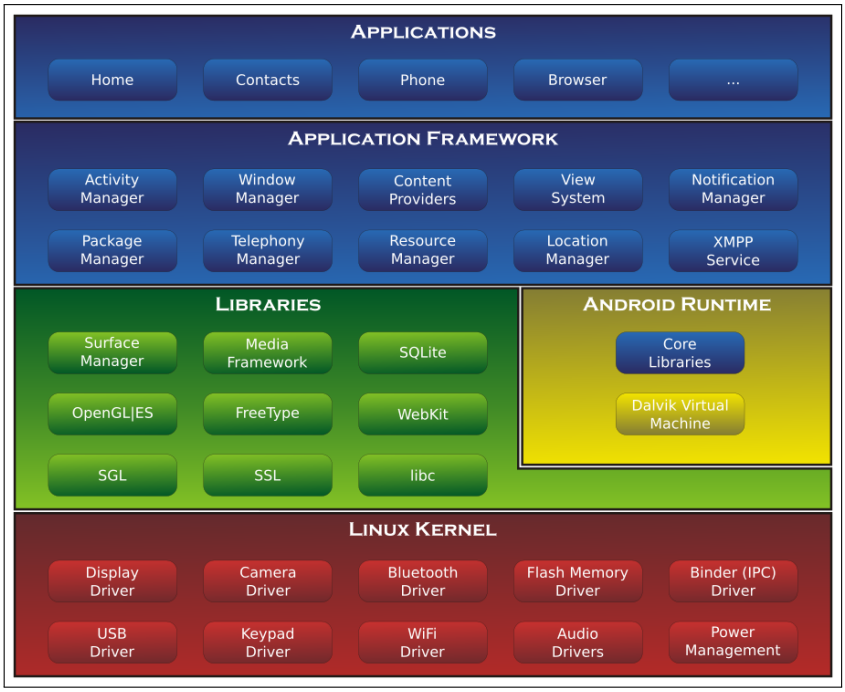

Figura 3.3: Arquitectura de capas de Android. Imagen obtenida de Wikipedia Android.

- **Applications:** Capa de alto nivel donde encontramos las aplicaciones del dispositivo.
- **Application Framework:** Capa de medio-alto nivel que le proporciona al desarrollador acceso completo a las mismas API´s del entorno de trabajo usadas por las aplicaciones base. La arquitectura está diseñada para la reutilización de componentes, por lo que capacidades de una aplicación pueden ser reutilizadas por otras.
- **Bibliotecas:** Capa de medio nivel en la que se incluyen un conjunto de librerías C/C++ usadas por varios componentes del sistema, como por ejemplo: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.
- **Android Runtime:** Capa de medio-bajo nivel que incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik.
- **Linux Kernel:** Capa de bajo nivel de la que Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. Además esta capa sirve de intermediaria entre el hardware y el software.

4 Análisis y especificación

4.1 Descripción general

En este capítulo se va a hablar de la fase de *Planteamiento de la app*, detallando información necesaria para el desarrollo de esta y para el entendimiento de cada apartado de la aplicación, así como de los términos utilizados.

En un primer lugar se mantuvieron varias reuniones de forma presencial, hasta la llegada del Covid-19, con el objetivo de entender que tipo de aplicación estaban buscando, dado que este proyecto es en colaboración con Departamento de Ciencias Ambientales y Recursos Naturales (DCARN), ya que leyendo la propuesta no era lo mismo buscar que la aplicación fuera más una herramienta abierta al publico, que el que esta aplicación estuviera orientada principalmente a los alumnos de la asignatura de Botánica, de la carrera de Biología, y abierta también al publico en general como herramienta para dar a conocer mejor el campus de la UA.

Llegados a la conclusión de que principalmente lo que mantenían era una propuesta de aplicación como diversión y herramienta de aprendizaje para los alumnos de la asignatura de Botánica, evitándoles así tener que realizar un cuaderno de clasificaciones sobre cada planta a papel, se determinó que un buen método para lograr lo propuesto era mediante la inclusión de componentes de gamificación. En un primer lugar se identificó que un método totalmente valido y útil para mantener la funcionalidad de la aplicación, como herramienta y diversión al mismo tiempo, sería el de efectuar una fotografía a la planta para realizar un reconocimiento de imagen mediante una red neuronal, dándole al usuario la posibilidad de que realizara el itinerario como una gymkana, teniendo que ir a la localización y buscar la planta en cuestión. Una vez buscada información y realizadas diferentes pruebas con redes neuronales, llegó en momento del estado de alarma debido al Covid-19, lo cual nos hizo cambiar totalmente nuestro propósito de añadir la red neuronal, ya que sería imposible poder realizar pruebas reales y recopilar una base de datos grande para poder tener la aplicación funcional en este aspecto, por lo que tras varias reuniones se les propuso a los profesores encargados del DCARN en el proyecto, si les parecía bien que el método a utilizar cambiara de forma que se podrían realizar una serie de preguntas al usuario, las cuales serían sobre apartados específicos de las plantas en cuestión, igualmente con la funcionalidad de una gymkana, de manera que el usuario debiera ir a la ubicación para encontrar la planta y contestar las preguntas. Como esta propuesta les agradó, se determinó que manera era la más óptima para realizar estas preguntas, por lo que ahí ellos nos propusieron utilizar las claves dicotómicas para las cuestiones.

Una vez determinado como se iban a realizar las preguntas, se debía llegar a un entendimiento de como serian estas, ya que entendíamos que la manera más eficaz para añadir estos itinerarios a la aplicación podía ser mediante ficheros .json descargados desde un servidor, permitiendo que los profesores encargados pudieran ir añadiendo itinerarios y estos fueran añadiéndose de forma automática al iniciar la aplicación. En este punto se concretó que cada

itinerario podía contener una serie de preguntas reutilizables, siempre en el mismo orden, para cada planta, las cuales se determinarían dentro de la aplicación como actividades, pero que cada una de estas tendría respuestas diferentes en cada cuestión planteada, permitiendo así identificar en cada momento la planta asociada a la actividad.

Como sabíamos que el objetivo de este proyecto era realizar una aplicación divertida y al mismo que funcionara como herramienta de aprendizaje y sustitutiva del cuaderno clasificación que utilizan en la asignatura de Botánica, les propuse a los profesores encargados del proyecto añadir un apartado de biblioteca de plantas, de manera que una vez cada usuario realizara una actividad de un itinerario, la planta asociada a esa actividad fuera desbloqueada de la biblioteca, por lo que se tendría otra pestaña añadida con todas las plantas, permitiendo así al usuario que pueda consultar siempre que lo desee esa planta en cuestión una vez la haya desbloqueado. Esto también sería un componente de gamificación, ya que el usuario podría ver la lista de plantas bloqueadas, y esto servirle de reto con el objetivo de querer ir desbloqueándolas completando actividades e itinerarios.

Llegados a este punto ya se ha concretado lo máximo posible como va a ser la estructura principal de la aplicación, la cual estará sujeta a cambios según se vaya avanzando en el desarrollo, pero que después de las diversas reuniones realizadas, esta estructura esta bastante trabajada. Para poder plasmarla se ha creado un *Mockup* que servirá de guía para realizar el diseño de la aplicación, tanto en iOS como en Android.

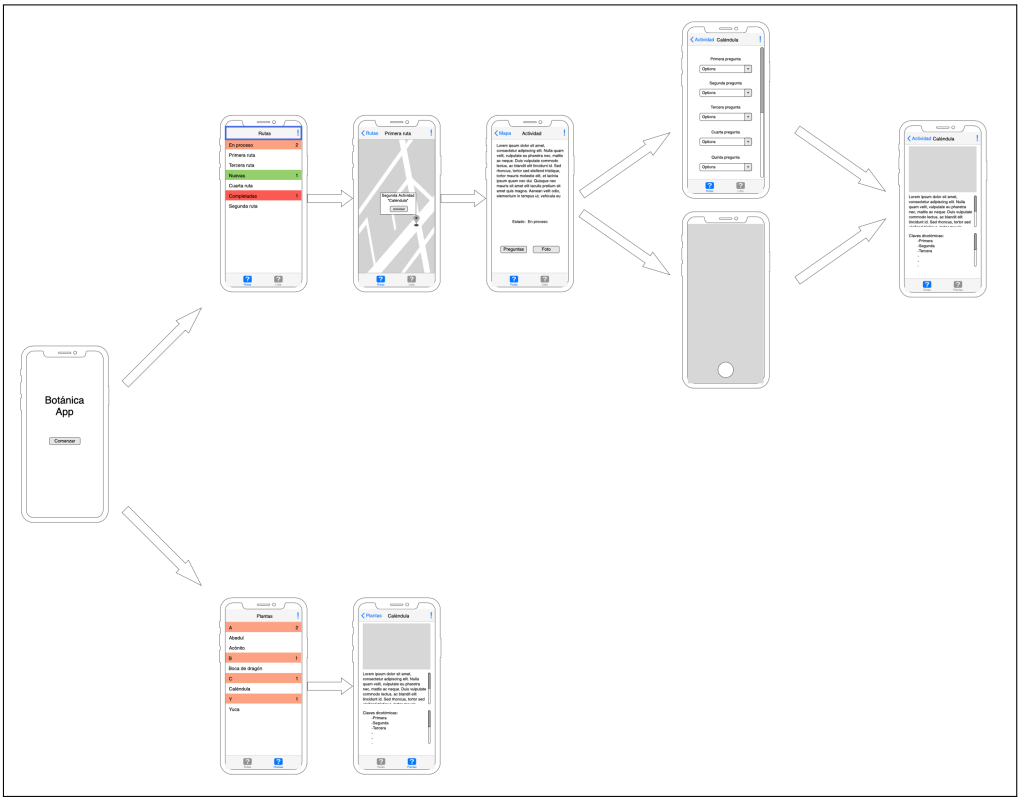

Figura 4.1: Mockup inicial de la aplicación.

Dado este *Mockup*, podemos distinguir las diferentes pestañas en orden jerárquico y los diferentes términos:

- **Pestañas**

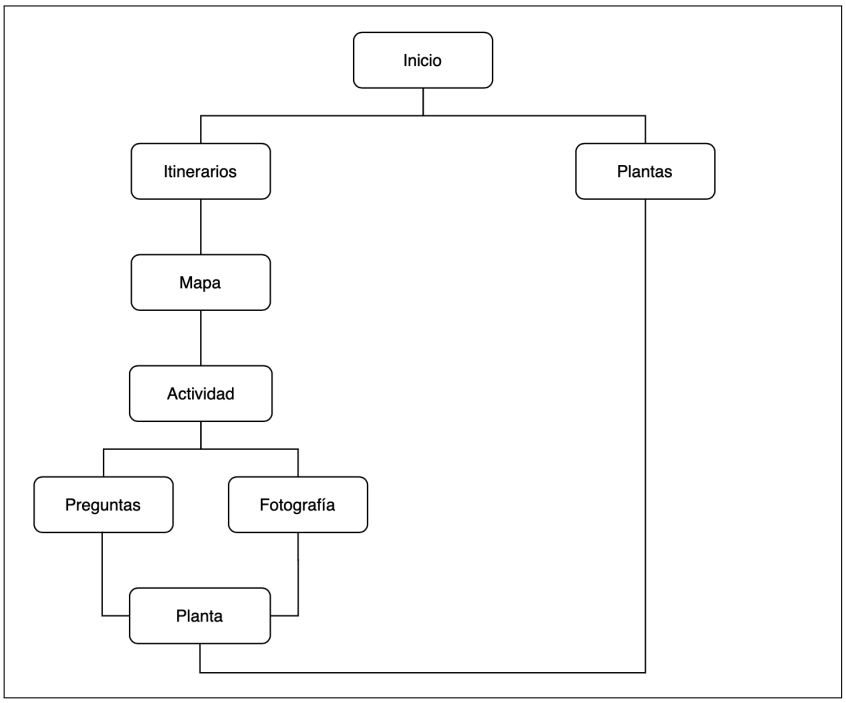

Figura 4.2: Estructura principal planteada inicialmente.

- **Términos**
  - **Itinerarios:** Rutas lineales con diferentes actividades ubicadas en un mapa, donde se tiene que ir contestando preguntas para desbloquear las plantas asignadas.
  - **Actividades:** Cada una de estas corresponde a una planta en una ubicación concreta, para ello se dispone de una fotografía e información de donde poder encontrarla. Para desbloquear la planta correspondiente hará falta contestar una serie de preguntas o sacar una fotografía a esta misma.
  - **Plantas:** Especies botánicas donde podemos encontrar información y fotografías sobre estas mismas. Para poder ver toda esta información se deberá completar la actividad correspondiente.
  - **Preguntas:** Cuestiones didácticas que se mostrarán con el objetivo de aprendizaje. Será necesario contestar correctamente todas las cuestiones asignadas a una actividad para desbloquear la planta asignada.
  - **Fotografía:** Captura sobre una planta concreta que deberá realizar el usuario con el objetivo de que la inteligencia artificial la reconozca y desbloqueé la planta [1].

[1] Funcionalidad no disponible por el Covid-19

Para poder crear todas estas secciones y guardar los datos en la app, se debe tener una base de datos con una estructura y unas clases principales. Según lo comentado, y sujeta a cambios según avance el desarrollo, la estructura inicial sería:

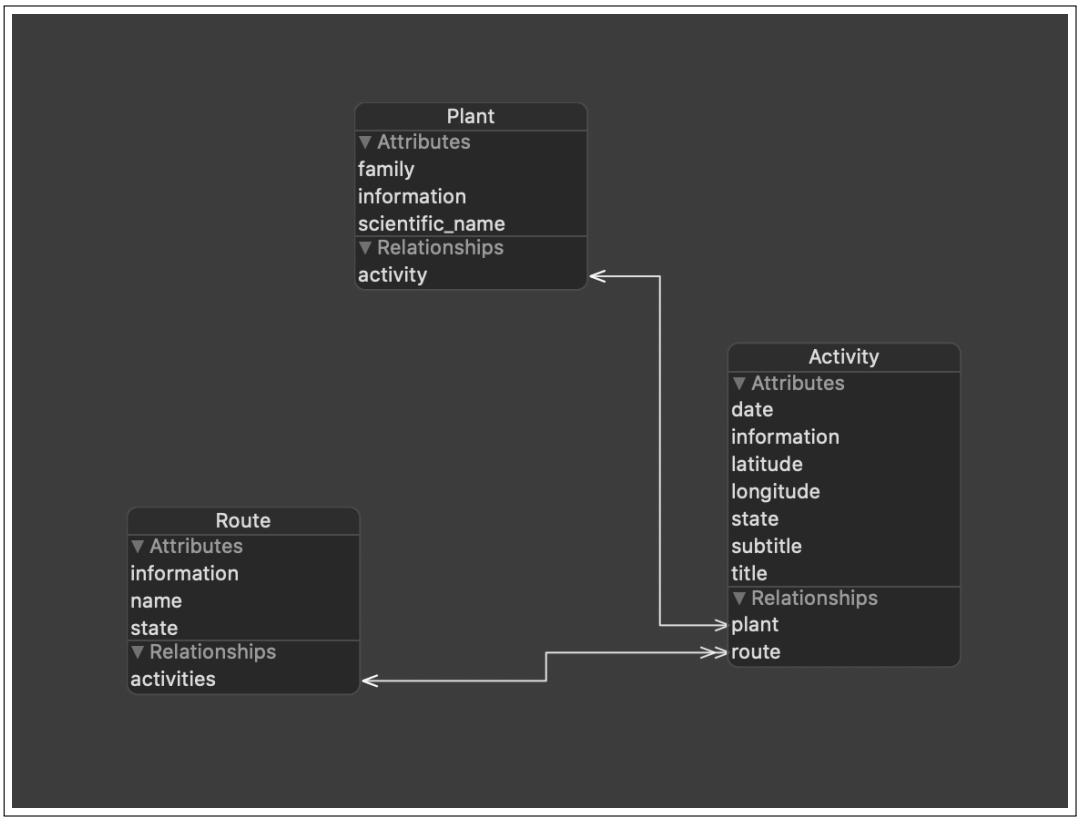

Figura 4.3: Planteamiento inicial del modelo de datos.

4.2 Funcionalidades de la aplicación

Podemos detallar las funcionalidades que permite la aplicación en la siguiente lista:

- **Descargar itinerarios:** Permite descargar los nuevos itinerarios añadidos al servidor.
- **Ver lista de itinerarios:** El usuario podrá ver la lista de itinerarios disponibles, en proceso y completado qué tiene descargos.
- **Ver lista de plantas:** El usuario podrá ver la lista completa de plantas asociadas a los itinerarios descargados.
- **Enviar un reporte de error:** En caso de que el usuario localice algún error, le permitirá enviar un email con el error.
- **Cargar mapa con las actividades:** Le mostrará al usuario un mapa interactivo de Google con todas las actividades marcadas en el mapa.

- **Ver detalle de una actividad:** El usuario podrá ver con detalle una actividad que seleccione en el mapa.
- **Completar una actividad:** Permite al usuario completar un test asociado a una actividad seleccionada, con el objetivo de desbloquear la planta asociada a esa.
- **Desbloquear una planta:** Si el usuario contesta todas las preguntas del test correctamente, le desbloqueará la planta de la lista de plantas.
- **Ver detalle de una planta:** El usuario podrá ver con detalle información relevante a la planta que seleccione en la lista de plantas.
- **Ver créditos:** Mostrará al usuario información relevante sobre la creación de la aplicación.

4.3 Especificación de requisitos

4.3.1 Requisitos funcionales

De forma ordenada según la pestaña a la que pertenecen, y de acuerdo con el estándar IEEE 830, aquí se recogen los requisitos funcionales que contiene la aplicación. Estos están divididos en dos tipos:

- **User Functional Requirement (UFR)**

Identificador del requisito	URF-01
Origen	Aplicación
Nombre	Lista de itinerarios
Descripción	Se muestra una lista de todos los itinerarios disponibles, los cuales están divididos en tres tipos: En proceso, Nuevos y Completados.

Identificador del requisito	URF-02
Origen	Aplicación
Nombre	Reporte de error
Descripción	Se abre la aplicación de correo electrónico que este disponible en el sistema, con el objetivo de que el usuario mande un e-mail con el suceso.

Identificador del requisito	URF-03
Origen	Aplicación
Nombre	Mapa de actividades
Descripción	Se muestra un mapa interactivo del itinerario seleccionado, donde el usuario puede localizar todas las actividades de este.

Identificador del requisito	URF-04
Origen	Aplicación
Nombre	Alerta de información del mapa
Descripción	Alerta que contiene información relevante del itinerario, además de una leyenda con el significado de los diferentes marcadores.

Identificador del requisito	URF-05
Origen	Aplicación
Nombre	Ventana de información de la actividad
Descripción	Al pulsar sobre un marcador se muestra una ventana con información previa de la actividad seleccionada, junto con un botón para ir a la actividad.

Identificador del requisito	URF-06
Origen	Aplicación
Nombre	Detalle de la actividad
Descripción	Se muestra una ventana modal con la información de la actividad, con las opciones de comenzarla o de contestar el test.

Identificador del requisito	URF-07
Origen	Aplicación
Nombre	Zoom fotografía
Descripción	Al pulsar sobre la imagen esta se amplía a pantalla completa, al volver a pulsarle se cierra.

Identificador del requisito	URF-08
Origen	Aplicación
Nombre	Formulario de preguntas
Descripción	Se muestra un formaulario de todas las preguntas y respuestas relacionadas con la actividad, las cuales se mostrarán en el orden que han sido añadidas.

Identificador del requisito	URF-09
Origen	Aplicación
Nombre	Ventana de error de las preguntas
Descripción	Al terminar el test y comprobar las respuestas, si alguna es errónea se muestra una ventana indicativa.

Identificador del requisito	URF-10
Origen	Aplicación
Nombre	Ventana de acierto de las preguntas
Descripción	Al terminar el test y comprobar las respuestas, si todas son correctas informa al usuario de que la planta asociada ha sido desbloqueada.

Identificador del requisito	URF-11
Origen	Aplicación
Nombre	Lista de plantas
Descripción	Se muestra una lista de todas las plantas disponibles, los cuales están divididas en secciones por orden alfabético.

Identificador del requisito	URF-12
Origen	Aplicación
Nombre	Alerta de error de la planta
Descripción	Alerta que se muestra en caso de que la planta no haya sido desbloqueada, mostrando la actividad que necesita completar para desbloquearla.

Identificador del requisito	URF-13
Origen	Aplicación
Nombre	Detalle de la planta
Descripción	Ventana que se abre a pantalla completa con toda la información relevante de la planta seleccionada.

Identificador del requisito	URF-15
Origen	Aplicación
Nombre	Lista de créditos
Descripción	Se muestra información relacionada con la creación de la aplicación.

- **System Functional Requirement (SFR)**

Identificador del requisito	SRF-01
Origen	Sistema
Nombre	Descarga de itinerarios
Descripción	Se realiza la conexión cliente-servidor y se comprueban los itinerarios disponibles en el servidor para su posterior descarga.

Identificador del requisito	SRF-02
Origen	Sistema
Nombre	Carga de itinerarios
Descripción	Se cargan todos los itinerarios almacenados en la Base de Datos (BBDD), y además se ordenan por orden alfabético.

Identificador del requisito	SRF-03
Origen	Sistema
Nombre	Lectura y muestra de actividades
Descripción	Se identifican todas las actividades de itinerario seleccionado, para posteriormente mostrarse en el mapa interactivo.

Identificador del requisito	SRF-04
Origen	Sistema
Nombre	Cambio de estado del itinerario
Descripción	Se leen los datos del itinerario en la BBDD y se cambia el valor del estado de este, guardando posteriormente el contexto para mantener la persistencia de datos.

Identificador del requisito	SRF-05
Origen	Sistema
Nombre	Cambio de estado de la actividad
Descripción	Se leen los datos de la actividad en la BBDD y se cambia el valor del estado de esta, guardando posteriormente el contexto para mantener la persistencia de datos.

Identificador del requisito	SRF-06
Origen	Sistema
Nombre	Cambio de estado de la planta
Descripción	Se leen los datos de la planta en la BBDD y se cambia el valor del estado de esta, guardando posteriormente el contexto para mantener la persistencia de datos.

Identificador del requisito	SRF-07
Origen	Sistema
Nombre	Carga de plantas
Descripción	Se cargan todas las plantas almacenadas en la BBDD, y además se ordenan por fecha de descarga.

4.3.1.1 Interfaz de usuario

La intención es de la que la interfaz de la aplicación sea lo más usable, sencilla e intuitiva posible, de manera que el usuario sea capaz de identificar rápidamente las funcionalidades y secciones de la misma. Después de realizar la búsqueda de aplicaciones relacionadas en el capítulo 2, se determinó utilizar una paleta de colores lo más viva y verde floral posible, pero manteniendo una correcta legibilidad para el usuario. Algunos de los colores que se pueden ver en la figura 4.4 se han utilizado en pequeños elementos.

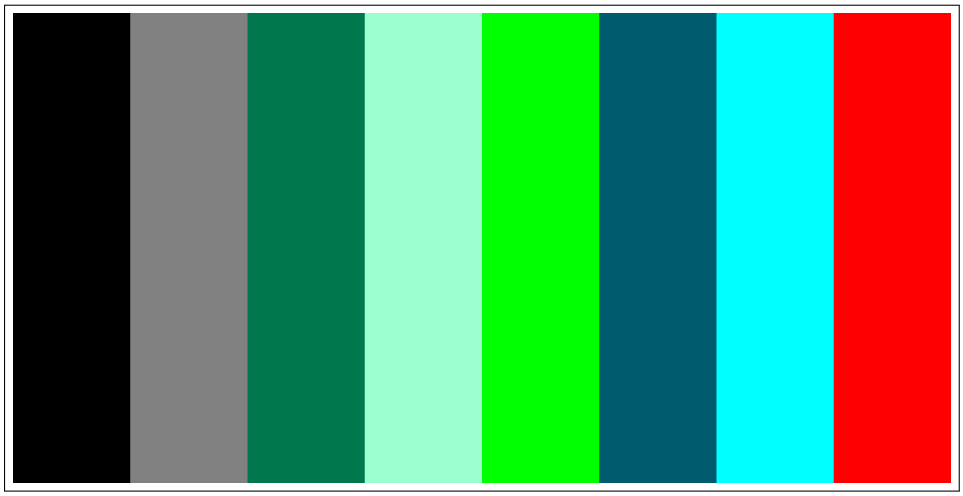

Figura 4.4: Paleta de colores utilizados en la aplicación.

4.3.2 Requisitos no funcionales

- **Usabilidad:** La aplicación debe ser sencilla e intuitiva de usar para todos los usuarios nuevos, ya que se busca que la gente la utilice para aprender botánica.

- **Disponibilidad:** Tanto la aplicación como el servidor deben funcionar constantemente para poder seguir descargando posibles nuevos itinerarios. Si el servidor no está disponible, la aplicación debe poder continuar funcionando correctamente con los itinerarios ya descargados.
- **Compatibilidad con plataformas:** Podrá ser instalada en cualquier dispositivo que cuente con el sistema operativo de iOS o Android, y al iniciar la descarga de nuevos itinerarios las peticiones desde ambos sistemas operativos se realizarán al mismo servidor.
- **Mantenibilidad:** Debe permitir que en futuras actualizaciones se puedan seguir añadiendo funcionalidades y corrigiendo errores.
- **Rendimiento:** La aplicación debe funcionar con fluidez y rapidez, ya que la carga de recursos debe ser baja, únicamente puede ser algo más pesada en la configuración del mapa.
- **Extensibilidad:** Debe permitir en un futuro añadir nuevas características de forma sencilla.
- **Protección medioambiental:** No utilizará ningún recurso externo para etiquetar plantas ni para la realización de nuevo itinerarios.
- **Documentación:** La aplicación debe tener material didáctico que explique como crear nuevos itinerarios o como modificarla en un futuro.

5 Desarrollo

5.1 Introducción

Ya hecha la propuesta inicial en el capítulo 4 de como va a ser inicialmente el proyecto, se da comienzo a la fase de desarrollo del proyecto, dividida a su vez en dos fases:

- Desarrollo iOS
- Desarrollo Android

Para la realización de este proceso, se ha seguido una metodología de tipo SCRUM, donde se han dividido las tareas en cuatro bloques: Nuevas, En proceso, Completadas y Para revisar.

5.2 Servidor y fichero JSON

El servidor está formado por diferentes rutas, permitiendo de esa manera acceder a cada itinerario de forma más rápida y concreta.

- **Servidor**
  - index.txt: Contendrá los nombre de todos los itinerarios, de manera que en la aplicación usaremos esos nombres como rutas de las carpetas y como nombres del .json.
  - Carpeta con el nombre del itinerario: Se dispondrá de una carpeta por cada itinerario. Esta carpeta será llamada con el nombre del itinerario, el cual almacenamos en el fichero "index.txt".
    - Archivo .json con toda la información del itinerario.
    - Imagen de la localización de cada actividad.
    - Imágenes del carrusel de cada actividad.

Código 5.1: Ejemplo de archivo index.txt del servidor

```
1 gimnospermas
```

Código 5.2: Ejemplo de archivo gimnospermas.json

```
{"itinerario":"Gimnospermas",
   "informacion_itinerario":"Informacion del itinerario",
   "plantas":[{
      "titulo_actividad":"actividad 1",
      "subtitulo_actividad":"Lugar donde esta o lo que queramos poner",
      "foto_localizacion":"localizacion.jpg",
      "informacion_actividad":"Informacion que queramos darle al usuario
          respecto a la actividad",
      "nombre_cientifico":"Cupressus sempervirens",
      "familia":"cupressaceae",
      "descripcion_planta":"Descripcion planta",
      "fotos_carrusel":"carrusel.jpg;carrusel2.jpg",
      "lat":"38.3864202198746",
      "lng":"-0.514705561099236",
      "preguntas":[{
         "titulo_pregunta":"H\u00e1bito",
         "respuestac":"Arbol",
         "respuestas":"arbusto con aspecto de palmera;Planta"
      }]
   }]
}
```

En el caso de este ejemplo citado de ”Gimnospermas”, la estructura del servidor seria:

- **Servidor**
  - index.txt: Con el nombre de ”Gimnospermas” en la primera linea, como podemos observar en el código 5.1
  - Gimnospermas
    - Gimnospermas.json
    - localizacion.jpg
    - carrusel.jpg
    - carrusel2.jpg

Ahora que ya tenemos determinada la estructura que usará el servidor, y la estructura que tendremos en el fichero .json, la cual fue ideada y pensada en parte en la fase de *Planteamiento de la app* como podemos ver en el capítulo 4, faltará automatizar la descarga de estos ficheros dentro de la aplicación, lo cual se realizará de manera concreta para cada sistema operativo. Para lograr esto, debemos conectarnos al servidor desde la url *”http://jtech.ua.es/uaplant/”*, la cual es la raíz desde la que posteriormente realizaremos las diferentes peticiones.

En primer lugar realizamos la petición al ”index.txt” desde el enlace *”http://jtech.ua.es/uaplant/index.txt”*, el cual parsearemos por ”\n”, ya que el nombre de cada itinerario está escrito en diferentes líneas, y estas están separadas mediante ese comando. Una vez ya hemos realizado la petición del archivo ”index.txt” y se ha parseado para obtener todos los nombres de los itinerarios disponibles, procedemos a realizar las peticiones a los diferentes itinerarios. Para realizar la petición del ejemplo del código 5.2, haremos uso de la url *”http://jtech.ua.es/uaplant/gimnospermas/gimnospermas.j* ya que ”gimnospermas” sería el nombre obtenido del ”index.txt”.

5.3 iOS

5.3.1 Visión general

En esta sección se va a concretar como se ha procedido al desarrollo de la aplicación de iOS de forma nativa. Para ello podemos observar como ha resultado el storyboard del desarrollo en la figura 5.1.

Además se han utilizado clases auxiliares para la realización de diferentes tareas:

- **Core Data:** Almacenamiento de datos en el dispositivo de forma persistente.
- **JSON:** Formato del fichero de lectura de los itinerarios del servidor, para lo que se ha implementado un lector especifico para estos datos.
- **CocoaPods:** Gestor de dependencias para XCode, el cual permite agregar frameworks de terceros a nuestro proyecto. Se ha usado para descargar: GoogleMaps, GooglePlaces, iOSDropDown (<jriosdev@gmail.com> (s.f.)) y FlexiblePageControl (shima11 (s.f.)).

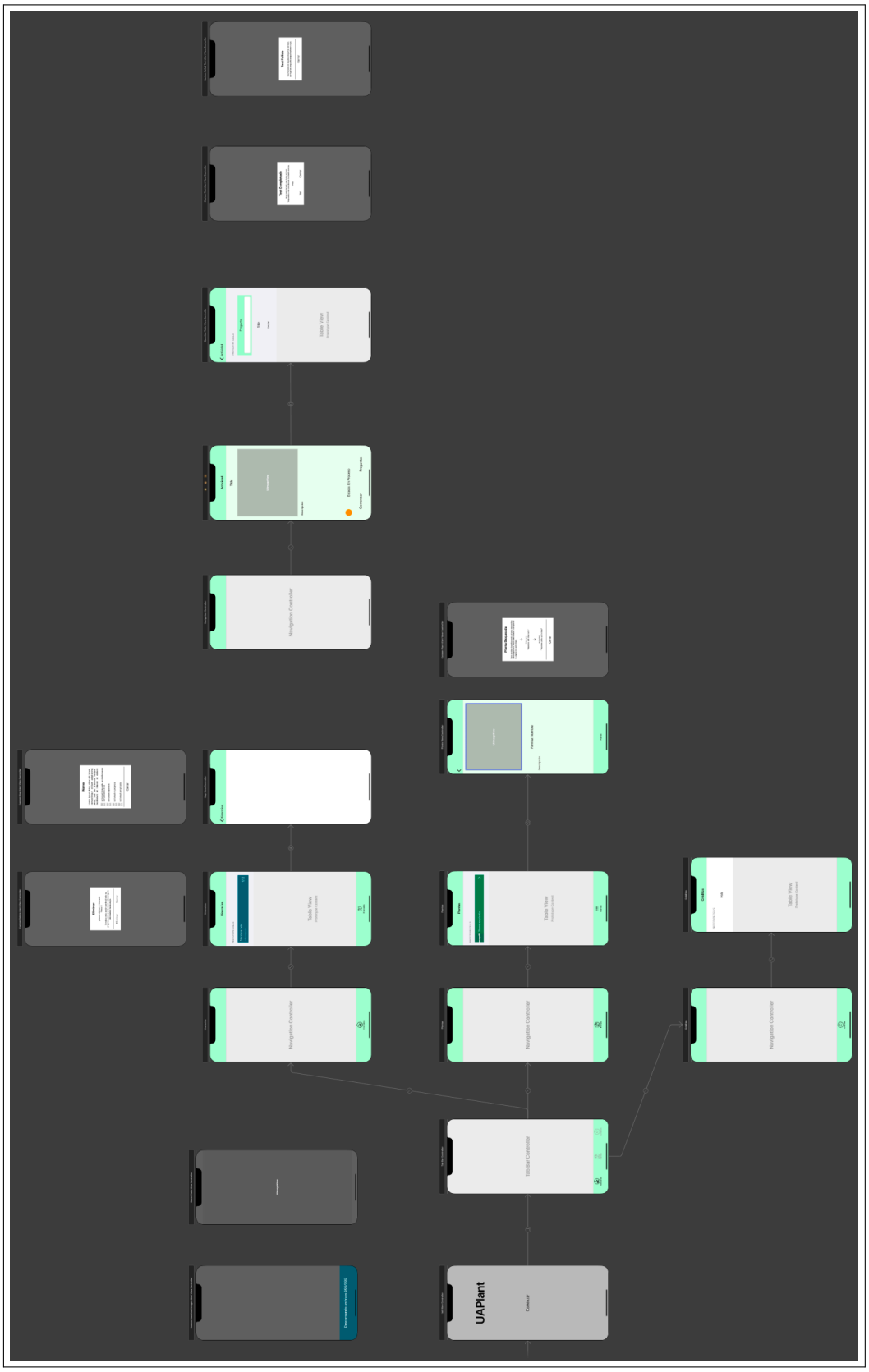

Figura 5.1: Storyboard resultante del desarrollo en XCode.

5.3.2 Creación del proyecto

En primer lugar el desarrollo de la aplicación para iOS pasa por la descarga y configuración del entorno de desarrollo *Xcode*, donde vamos a realizar la programación para este sistema operativo de forma nativa al completo.

Posteriormente a la descarga de Xcode, creamos el proyecto con un nombre clave, en este caso al ser el desarrollo para el TFM se le pondrá como nombre *"TFM-Botanica"*, activando el uso de Core Data en la creación de este.

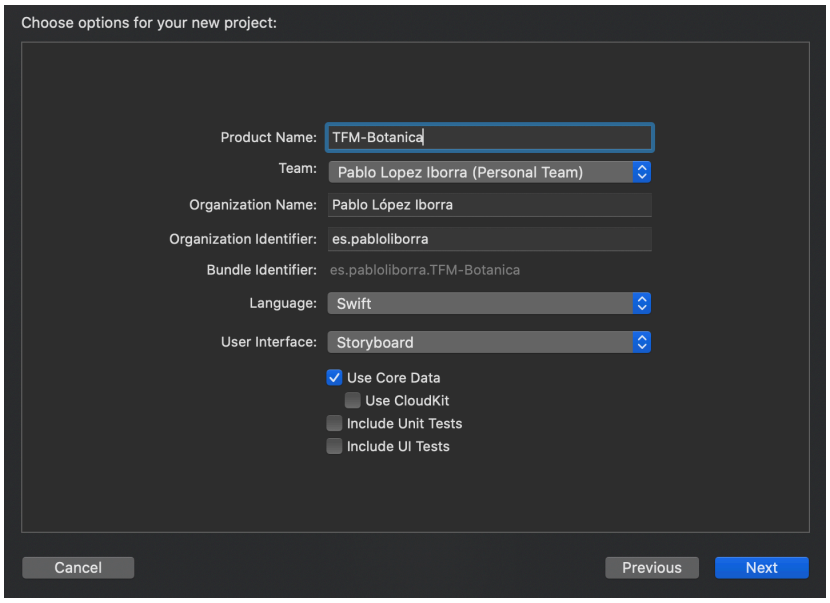

Figura 5.2: Creación del proyecto iOS.

Para poder comenzar a crear partes de la aplicación, deberemos crear en Core Data las entidades necesarias para almacenar los datos. A su vez estas entidades tendrán atributos que las conformarán, como podemos ver en la Figura 4.3. Las entidades que crearemos en Core Data corresponderán a cada tabla necesaria, donde se almacenarán todos los datos, como son:

- Itinerario -> Route
- Actividad -> Activity
- Planta -> Plant

Dado que el proceso de desarrollo acaba de comenzar, es muy probable que a estas entidades se les sumen otras, ya que pensando a posteriori, es probable que nos hagan faltas entidades para: preguntas, respuestas y quizás para imágenes, según vayamos a poder tener una o varias en otras entidades.

5.3.3 Pantalla inicial

Se añade una primera pantalla de bienvenida al usuario, la cual posteriormente servirá para la descarga de itinerarios o rutas en segundo plano.

Figura 5.3: Primera instancia de la pantalla inicial.

Una vez ya ha cargado la aplicación, sabemos que deseamos tener dos secciones en la pantalla principal, por lo que de forma consiguiente a la pantalla principal de bienvenida, añadimos un "Tab Bar Controller" y a este a su vez un "Navigation Controller" para poder gestionar los diferentes niveles de nuestras pantallas:

- **Itinerarios:** Lista de los itinerarios.
- **Plantas:** Lista de plantas, correspondientes a las actividades de los itinerarios.

5.3.4 Pantalla de lista de itinerarios

Una vez ya tenemos la distinción entre diferentes pantallas principales, en primer lugar la pantalla de ”Itinerarios” la creamos con un controlador de tipo ”Table View Controller” que nos permitirá listar todos los itinerarios divididos en tres secciones diferentes. Estas secciones corresponderán al estado actual de los itinerarios:

- Rutas disponibles
- Rutas en progreso
- Rutas completadas

Para poder listar los itinerarios de esta manera, y tener una ejecución más simple, creamos un nuevo Cocoa Touch Class de tipo ”Table View Controller”, donde se añaden tres arrays de tipo *”Route”*, los cuales corresponderán a cada una de las secciones dichas anteriormente. En este controlador es donde se configurará todo lo principal correspondiente a la lista de itinerarios.

Código 5.3: Arrays de itinerarios

```
class RouteTableViewController: UITableViewController {

    let sections = ["En proceso", "Nuevas", "Completadas"]
    var routesAvailables: [Route] = []
    var routesInProgress: [Route] = []
    var routesComplete: [Route] = []
```

Para poder crear una lista con unas celdas a nuestro gusto, deberemos añadir un ”Table View Cell” y cambiar en el Storyboard que nuestra celda haga uso de este controlador. Una vez hecho eso, creamos la celda a nuestro gusto, dejándola con una apariencia final como podemos ver en la figura 5.4.

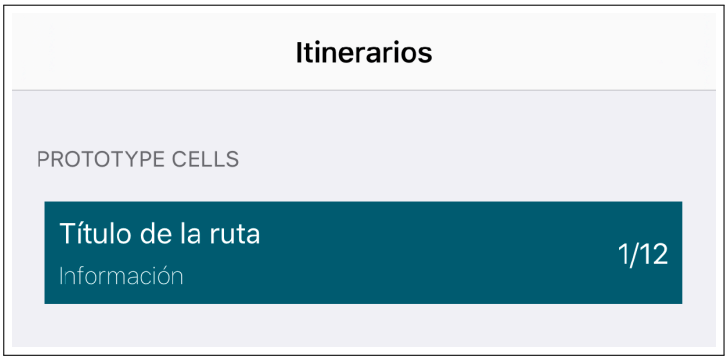

Figura 5.4: Celda final de la lista de itinerarios.

En esta celda podemos distinguir diferentes puntos:

- Título de la ruta
- Información breve sobre la ruta
- Número de actividades completadas/totales que tiene el itinerario

5.3.5 Pantalla de lista de plantas

Una vez hemos realizado la configuración de la celda para la lista de itinerarios, procedemos a realizar lo mismo con la lista de plantas.

Creamos un "Table View Controller" donde gestionaremos todas las plantas que tengamos en los itinerarios. Esta gestión se realizará por orden alfabético y divididas en secciones según su primera letra, y para conseguir eso deberemos gestionar las plantas que tengamos guardadas en Core Data mediante un diccionario y un array de las secciones:

Código 5.4: Gestión de la lista de las plantas guardadas en Core Data

```
var plantsDictionary = [String: [Plant]]()
var plantSectionTitles = [String]()

func loadPlantsCoreData() {
    guard let miDelegate = UIApplication.shared.delegate as? AppDelegate else ↩
        ↪ {
        return
    }
    let miContexto = miDelegate.persistentContainer.viewContext

    let requestPlants : NSFetchRequest<Plant> = NSFetchRequest(entityName:"↩
        ↪ Plant")
    var plants = try? miContexto.fetch(requestPlants)

    plants!.sort(by: {$0.scientific_name!.lowercased() < $1.scientific_name!.↩
        ↪ lowercased()})

    for plant in plants! {
        let plantKey = String(plant.scientific_name!.prefix(1))
        if var plantValues = plantsDictionary[plantKey] {
            plantValues.append(plant)
            plantsDictionary[plantKey] = plantValues
        } else {
            plantsDictionary[plantKey] = [plant]
        }
    }

    plantSectionTitles = [String](plantsDictionary.keys)
    plantSectionTitles = plantSectionTitles.sorted(by: { $0 < $1 })
}
```

Posteriormente a esto, procedemos a configurar la celda que se mostrará en la lista de plantas, la cual queda finalmente de esta manera:

Figura 5.5: Celda final de la lista de plantas.

En esta celda podemos distinguir diferentes puntos:

- Imagen de previsualización de la planta
- Nombre de la planta

5.3.6 Mapa con actividades

Una vez ya se han configurado ambas pestañas de forma inicial, tanto la de itinerarios como la de plantas, procedemos a añadir la pestaña de "Mapa" dentro de cada itinerario.

Como se pretende que tanto la aplicación de iOS como la de Android sean lo más similares posibles, aquí procedemos a añadir como mapa el de Google Maps, ya que la api está disponible para ambos sistemas operativos. Para poder usar este mapa deberemos instalar los pods "pod 'GoogleMaps'" y "pod 'GooglePlaces'" mediante CocoaPods.

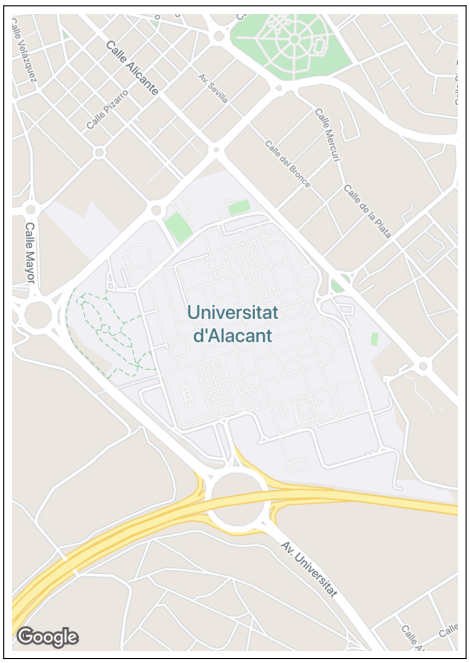

Figura 5.6: Google Maps como mapa principal para mostrar las actividades.

Una vez añadido ya el mapa, el siguiente paso seria configurar las actividades que deben mostrarse en cada mapa correspondiente a cada actividad. Para ello, como hemos añadido una relación de ”uno a muchos” entre los itinerarios y las actividades, tenemos todas las actividades asociadas a cada itinerario, por lo que debemos cargarlas y añadirlas en el mapa según sus coordenadas. Además siempre queremos saber el estado de la actividad, para saber si esta:

- Disponible
- En proceso
- Completada
- Inactiva

Para poder gestionar eso, creamos una clase nueva llamada ”CustomMarker”, la cual gestionara los tipos de estados que tenemos (States), y el estado actual de cada marcador según el estado de la actividad. Según el estado en el que se encuentre la actividad, el marcador será de un color diferente, por lo que al inicializador le pasaremos el estado actual, y añadiremos además un setState para cambiarlo en caso necesario.

Código 5.5: Gestión de la lista de las plantas guardadas en Core Data

```
struct State {
    //CONSTANTS: State of the marker
    public static let IN_PROGRESS = 0
    public static let AVAILABLE = 1
    public static let COMPLETE = 2
    public static let INACTIVE = 3
}

class CustomMarker: GMSMarker {
    var state:Int = -1

    init(state: Int) {
        super.init()

        self.state = state

        switch state {
            case State.IN_PROGRESS:
                self.icon = GMSMarker.markerImage(with: UIColor.red)
            case State.AVAILABLE:
                self.icon = GMSMarker.markerImage(with: UIColor.cyan)
            case State.COMPLETE:
                self.icon = GMSMarker.markerImage(with: UIColor.green)
            case State.INACTIVE:
                self.icon = GMSMarker.markerImage(with: UIColor.gray)
                self.isTappable = false
            default:
                break
        }
```

```
    }
    public func setState(state: Int) {
        self.state = state
        switch state {
            case State.IN_PROGRESS:
                self.icon = GMSMarker.markerImage(with: UIColor.red)
                self.isTappable = true
            case State.AVAILABLE:
                self.icon = GMSMarker.markerImage(with: UIColor.cyan)
                self.isTappable = true
            case State.COMPLETE:
                self.icon = GMSMarker.markerImage(with: UIColor.green)
                self.isTappable = true
            case State.INACTIVE:
                self.icon = GMSMarker.markerImage(with: UIColor.gray)
                self.isTappable = false
            default:
                break
        }
    }
}
```

5.3.7 Ventana con significado del marcador

Una vez añadidos los marcadores de cada actividad correspondiente, creamos la ventana de información que vamos a mostrar cuando pulsemos sobre un marcador. Para crear la ventana, creamos un nuevo layout.xml con el diseño que queremos que tenga.

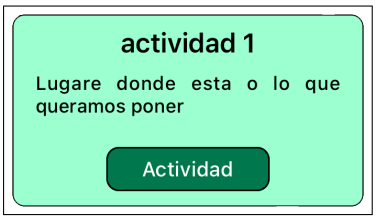

Figura 5.7: Ventana de información del marcador seleccionado.

5.3.8 Ventana informativa con significado de los marcadores

Para aclararle al usuario el significado de los colores de los ”markers” y los estados, hemos añadido una pantalla informativa a la que se accederá mediante un botón a la derecha en la barra de navegación.

Figura 5.8: Botón de información de los ”markers” en la barra de navegación.

Figura 5.9: Pantalla informativa de los "markers" y los estados de las actividades.

5.3.9 Remodelación del modelo de datos

Llegados a este punto, e iterando sobre lo que se planteó en primera instancia, es necesario realizar una nueva revisión al modelo de datos que tenemos en Core Data, ya que para poder realizar las siguientes pestañas de información de la actividad y las propias preguntas del test, es necesario añadir nuevas entidades a las ya creadas, como son:

- Imagen -> Image
- Pregunta -> Question
- Respuesta -> Answer

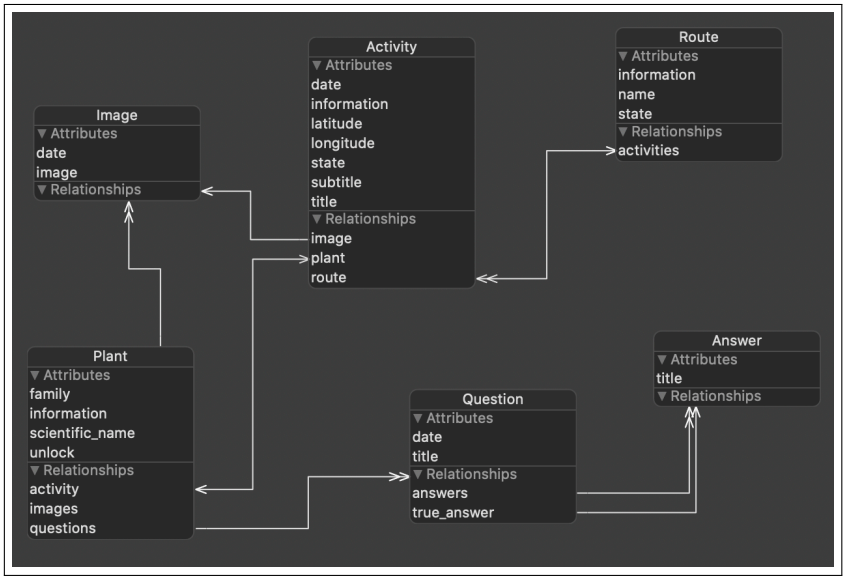

Figura 5.10: Base de datos resultante de una nueva iteración sobre la figura 4.3.

5.3.10 Pantalla con la información de la actividad

Una vez tenemos en nuevo modelo de datos, creamos las entidades correspondientes, y procedemos a crear un nuevo controlador de tipo "View Controller" donde se va a mostrar la información de la actividad. Esta pestaña se mostrará de forma modal al pulsar sobre la información de un marcador del mapa, y contendrá diferentes datos como son:

- Título de la actividad
- Fotografía de la localización
- Descripción de la actividad
- Color del estado de la actividad
- Estado de la actividad
- Botones
  - Comenzar actividad
  - Preguntas del test

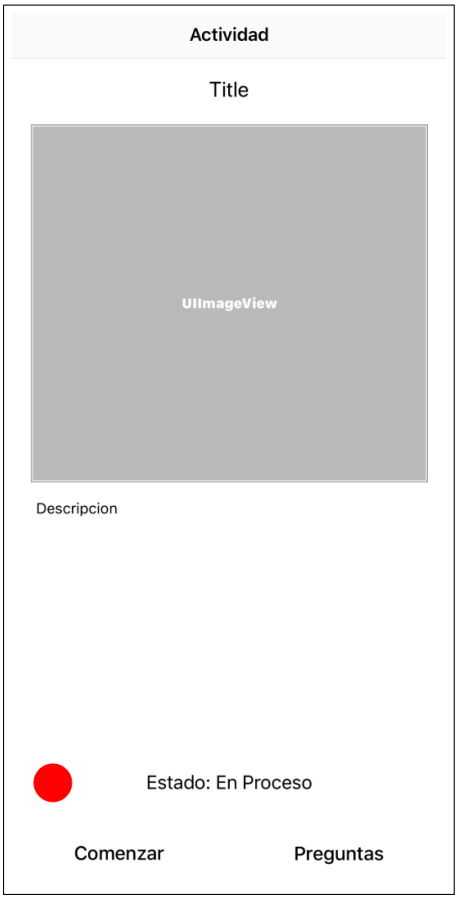

Figura 5.11: Pestaña de información sobre la actividad pulsada.

Al darle al botón de comenzar cambiaremos el estado de la actividad, que pasará de ”Disponible” a ”En Proceso”, guardando este cambio automáticamente en Core Data, de manera que si la actividad que acabamos de comenzar es la primera del itinerario elegido, el propio itinerario también cambiará de estado, de ”Disponible” a ”En Proceso”, actualizando a su vez el ”marker” del mapa. Podemos ver el flujo del cambio de estado en la figura 5.12.

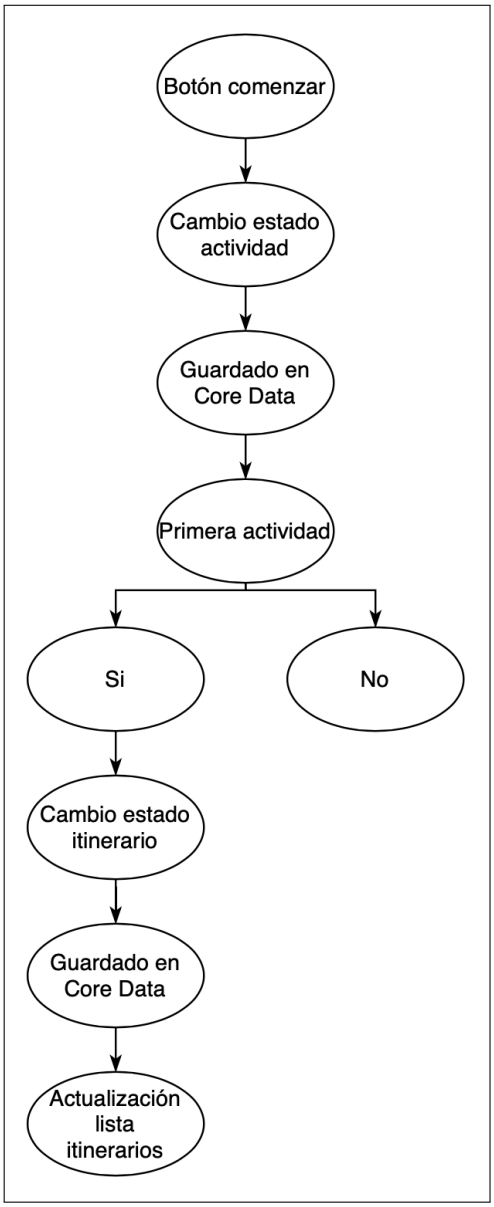

Figura 5.12: Diagrama de flujo de cambio de estado de la actividad.

Una vez ya se ha realizado el cambio de estado y se han comprobado el resto de estados, como se muestra en el diagrama de flujo, procedemos a bloquear el botón de comenzar actividad y a desbloquear el botón que permite realizar el test.

5.3.11 Pantalla con la lista de preguntas y respuestas del test

Para realizar el test, creamos un ”Table View Controller” que gestionará todos los datos y las celdas de las diferentes preguntas asociadas a la actividad. Creamos dos ”Table View Cell”, el primero para darle forma a la celda de las preguntas, ya que la intención es poder mostrar el título de la pregunta y un desplegable con todas las respuestas asociadas a esa pregunta, y el segundo ”Table View Cell” que creamos es para la última celda, la cual contendrá un botón ”Enviar” que comprobará si todas las respuestas son correctas o no, y abrirá un panel correspondiente a cada caso.

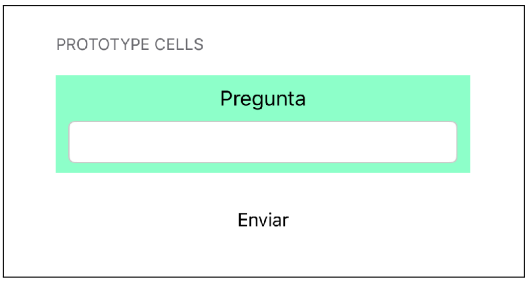

Figura 5.13: Celdas disponibles en el controlador de las preguntas del test.

Para poder realizar el desplegable de las respuestas de cada pregunta, se ha importado un nuevo CocoaPod llamado ”pod ’iOSDropDown’”, el cual se trata de un desplegable que permite añadir tantas casillas como se deseen.

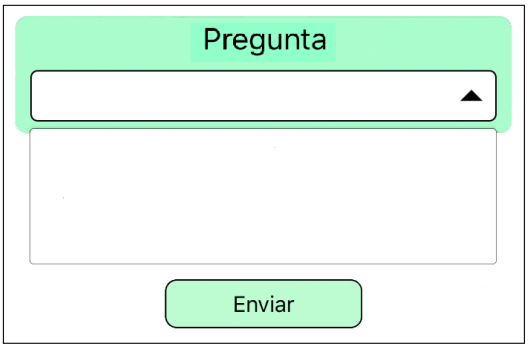

Figura 5.14: DropDown para las respuestas del test.

5.3.12 Ventanas de información sobre el resultado del test

Una vez ya tenemos la configuración de las celdas del test, habría que realizar la configuración para la corrección de estas preguntas y respuestas, de manera que una vez el usuario pulse el botón de ”Enviar”, cambiaremos el color de fondo del dropdown de manera que le proporcione al usuario un feedback de que respuestas son correctas y cuáles no. Para que el usuario comprenda el código de colores elegido para las respuestas correctas y las incorrectas, en caso de tener alguna respuesta fallida, se le mostrará un mensaje al usuario indicándole que ha pasado y lo que debe corregir.

Figura 5.15: Alerta indicativa de que el usuario ha fallado alguna respuesta.

Como podemos observar en la figura 5.15, indicamos al usuario que las respuestas marcadas en rojo son incorrectas, y que debe corregirlas para poder finalizar el test. Llegados a ese punto, si el usuario finaliza correctamente el test, se le mostrará otra alerta indicativa diciéndole que ha logrado desbloquear la planta correspondiente, y que tiene la opción de continuar completando actividades o de ir a ver la información de la planta.

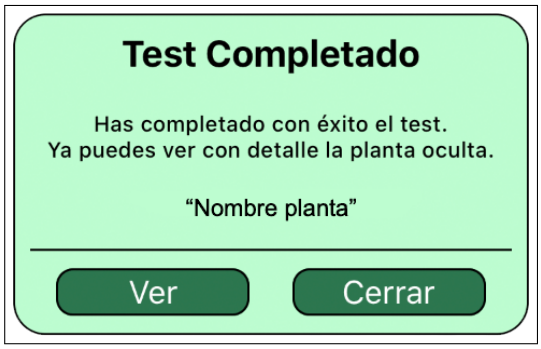

Figura 5.16: Alerta indicativa de que el usuario ha finalizado el test correctamente.

Una vez la planta se ha desbloqueado, cambiaremos en Core Data el atributo ”unlock” de la entidad ”Plant”, como podemos ver en la figura 5.10. Este atributo fue añadido para obligar al usuario a realizar cada actividad correspondiente si este deseaba poder consultar la planta en la lista de estas, por lo que una vez que se ha completado el test, realizamos lo comentado, guardamos este cambio en Core Data con el objetivo de que el usuario ya disponga de esta planta para futuras consultas.

5.3.13 Pantalla con la información de la planta

Ahora que ya se ha completado el flujo de los itinerarios, las actividades, las preguntas y el desbloqueo de la planta, procedemos a realizar la pantalla de información de la planta, de manera que si la desbloqueamos y pulsamos ”Ver”, como podemos observar en la figura 5.16, o si seleccionamos la planta deseada de la lista de estas debe mostrarnos:

- Nombre de la planta
- Carrusel de imágenes

- Familia a la que pertenece
- Información relevante

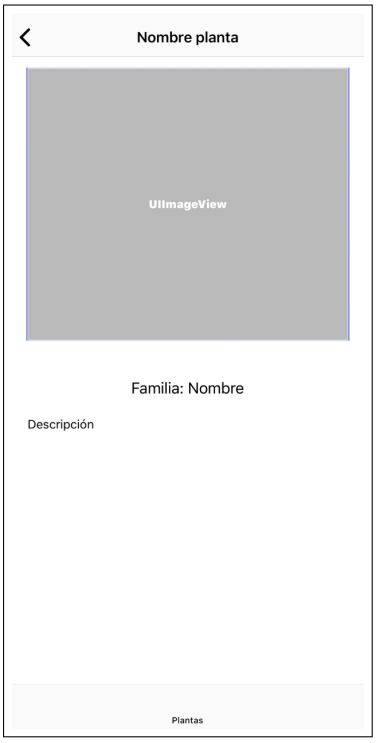

Figura 5.17: Pantalla de información sobre una planta.

Cabe destacar que en el caso de que el usuario desee información sobre una planta bloqueada, se le mostrará una alerta indicativa de que actividad debe completar para desbloquearla.

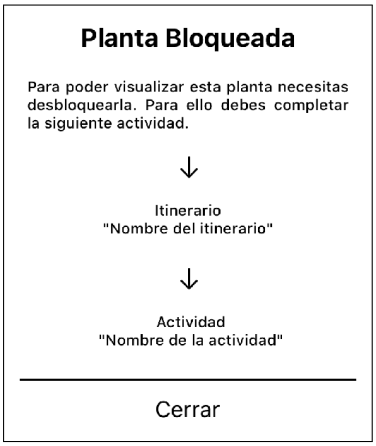

Figura 5.18: Alerta indicativa de que la planta seleccionada esta bloqueada y que actividad debe completar para desbloquearla.

5.3.14 Pantalla de Créditos y cambio de diseño e iconos de la aplicación

Ahora que ya hemos completado el flujo total de la aplicación, se va a proceder a iterar sobre los colores, fondos y diseños de la aplicación, utilizando como colores principales:

- Celda lista itinerarios -> #235B6E
- Celda lista plantas -> #2D764F
- Botones principales -> #2D764F
- Celda lista preguntas -> #BEFFD1
- Barras de navegación -> #BEFFD1
- Fondo de las alertas -> #BEFFD1

Se añade además una nueva pestaña adicional "Créditos" a nuestro "Tab Bar Controller" inicial, de manera que una vez añadida esta nueva pestaña, y cambiados los colores y los iconos de las pestañas, el resultado seria:

Figura 5.19: Tab Bar final de la pagina principal.

Configuramos los créditos mediante un nuevo "Table View Controller" y un "Table View Cell" que nos permitirá añadir las secciones que deseemos:

- Desarrollador
- Colaboraciones
- Propuesta
- Recursos externos

Cambiamos el diseño de las secciones de la lista de itinerarios y de la lista de plantas, de manera que para ambos casos se ha creado un nuevo archivo .xib con el diseño deseado. En el caso del diseño de las secciones de la lista de itinerarios se ha configurado de manera que la primera sección muestra el total de itinerarios completados/disponibles, y en el resto de secciones disponibles se muestra una imagen, el título y el número de itinerarios que contiene, esto lo podemos ver en la figura 5.20. Por otro lado, el diseño elegido para las secciones de la lista de plantas es mostrar en la primera el total de plantas desbloqueadas/disponibles, y en el resto de secciones se muestra el título y la lista de plantas que contiene, esto lo podemos observar en la figura 5.21.

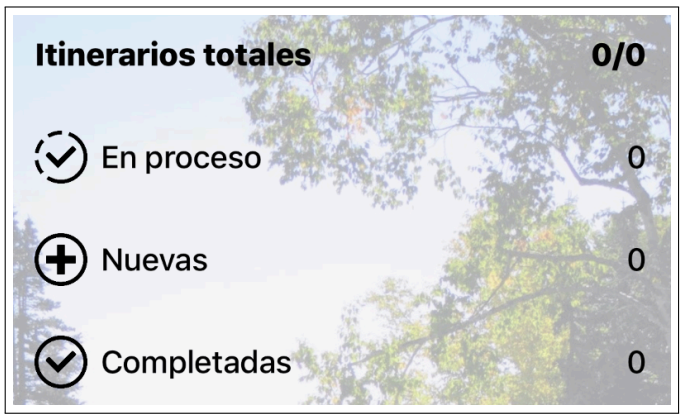

Figura 5.20: Diseño de las secciones de la lista de itinerarios.

Figura 5.21: Diseño de las secciones de la lista de plantas.

5.3.15 Conexión y lectura de datos del servidor

Como ya se ha explicado en la sección 5.2 de "Servidor y fichero JSON", realizaremos la conexión al servidor y parsearemos el JSON con la información de los itinerarios. Para realizar estas peticiones en Swift utilizamos el comando:

Código 5.6: Ejemplo de petición de datos al servidor mediante URLSession

```
URLSession.shared.dataTask(with: urlIndex) { data, response, error in
  //Data contiene los datos obtenidos del servidor.
}.resume()
```

Cuando ya hemos realizado la petición de descarga del fichero .json, en primer lugar comprobaremos que ese itinerario no existe previamente en la Core Data. De existir previamente en la memoria no se volvería a guardar, pero si este no existe debemos parsearlo para obtener los diferentes datos del itinerario, de manera que conforme se vayan obteniendo, se irán guardando en Core Data creando las entidades necesarias. Un ejemplo de como se guardaría en Core Data la información de la entidad "Route" seria:

Código 5.7: Ejemplo de guardado de los datos de la entidad "Route"

```
let routeData = NSEntityDescription.insertNewObject(forEntityName: "Route", ↩
    ↪ into: miContexto) as? Route
routeData?.name = route
if let information = json["informacion_itinerario"] as? String {
   routeData?.information = information
```

```
}
routeData?.state = Int16(State.AVAILABLE) //Estado inicial siempre "Available
    ↪ ", ya que todas las rutas estarán disponible una vez guardadas.
```

5.3.16 Gestor de descarga

Continuando con los datos extraídos del .json, una vez se llegue al apartado de las imágenes, crearemos un gestor de descargas, el cual comenzará la descarga de las imágenes de forma inmediatamente posterior al guardado de todos los datos necesarios de las diferentes entidades asociadas al itinerario. Para que este gestor pueda realizar las descargas de las diferentes imágenes, mientras se vayan guardando todos los datos iremos guardando en dos diccionarios diferentes la información necesaria para ello. Por un lado tendremos un primer diccionario que almacenará la url de la imagen de la localización de la actividad, y la propia actividad donde guardarla, y por otro lado tendremos otro diccionario que almacenará las urls de las diferentes imágenes del carrusel de la planta, y la propia planta donde almacenar esas imágenes.

Código 5.8: Diccionarios donde se almacenan las urls de las imágenes a descargar en el gestor

```
static var imagesLocalizationToDownload = [Activity: String]()
static var imagesPlantsToDownload = [Plant: [String]]()
```

Una vez que ya se han descargado todas las imágenes necesarias desde el gestor de descargas, guardaremos el contexto con todos los datos de los diferentes itinerarios añadidos en Core Data. Para poder realizar el guardado del contexto actual desde el gestor, en el cual no tenemos acceso al contexto abierto previamente, lo haremos mediante la función del código 5.9, la cual permite guardar el contexto general de la aplicación.

Código 5.9: Guardado del contexto de la aplicación desde una función sin acceso al contexto abierto previamente

```
(UIApplication.shared.delegate as! AppDelegate).saveContext()
```

5.3.17 Pantalla y banner de descarga de archivos

Ahora que ya se han configurado las diferentes peticiones al servidor, procedemos a añadir una pantalla de descarga que avisará al usuario de que se están descargando los diferentes itinerarios del servidor. Para mostrarle al usuario correctamente esta pantalla de descarga, en cada momento habrá que diferenciar los dos posibles lugares desde los que se podrán realizar estas descargas:

- **Pantalla inicial de bienvenida:** Se mostrará a pantalla completa un diálogo que especificará los archivos descargados/totales a descargar. Podemos ver un ejemplo de esta pantalla en la figura 5.22.
- **Pantalla de la lista de itinerarios:** Aparecerá un banner con un diálogo que especificará los archivos descargados/totales a descargar. Podemos observar un ejemplo este banner en la figura 5.23.

Una vez ya se han añadido los informativos de las descargas que se están realizando, solo faltaría añadir cuatro últimas cosas:

Figura 5.22: Pantalla de descarga de archivos en la pantalla de bienvenida de la aplicación.

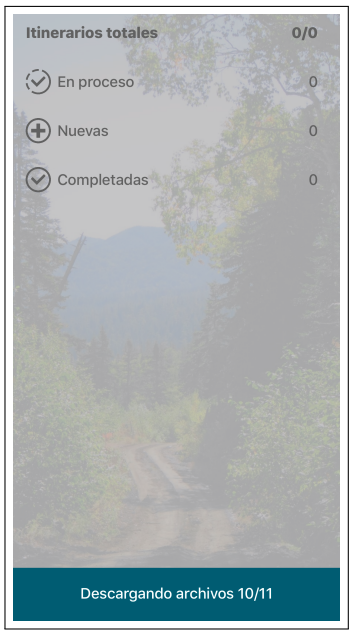

Figura 5.23: Banner de descarga de archivos en la lista de itinerarios.

- Botón para realizar la descarga de archivos desde la lista de itinerarios.
- Borrado de un itinerario.
- Botón para avisar de un posible error en la aplicación.
- Enviar aviso de error por correo.

Añadimos el botón para realizar la descarga de archivos desde la lista de itinerario, de manera que la barra de navegación tendría el siguiente aspecto:

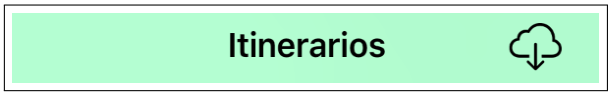

Figura 5.24: Barra de navegación de la lista de itinerarios con el botón para realizar las descargas del servidor.

5.3.18 Ventana de información del borrado de un itinerario

Para borrar un itinerario disponible, añadimos en nuestro ”Table View Controller” de la lista de rutas, que al realizar una pulsación larga abra una alerta que nos dará a elegir si queremos borrar este itinerario o no. En caso de seleccionar ”Eliminar”, realizaremos una búsqueda en Core Data, sobre la entidad ”Route”, con el nombre del itinerario.

Código 5.10: Búsqueda de un itinerario en Core Data para su posterior eliminación

```
guard let miDelegate = UIApplication.shared.delegate as? AppDelegate else {
    return
}
let miContexto = miDelegate.persistentContainer.viewContext

let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: "Route")
let predicateName = NSPredicate(format: "name == %@",self.route!.name!)
fetchRequest.predicate = predicateName
```

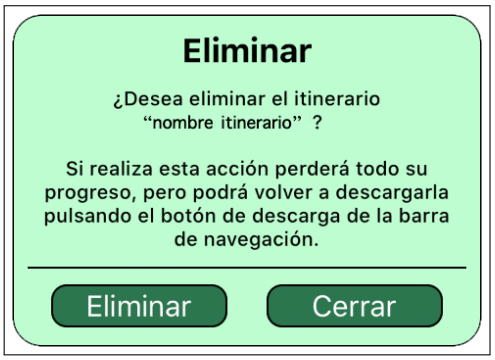

Figura 5.25: Alerta de borrado de un itinerario

5.3.19 Envío de una notificación de error mediante correo electrónico

Por ultimo, para finalizar la aplicación, faltaría añadir el botón para avisar de un error y la funcionalidad de este mismo enviando un correo. El botón para abrir el gestor de correo electrónico, lo debemos añadir en todas las pestañas de nuestra aplicación, dándole la posibilidad al usuario de reportar un posible error en cualquier punto de esta. A este botón a su vez deberemos añadirle la funcionalidad de que abra el gestor de correo electrónico, por lo que le añadiremos una llamada a otra función y clase que realice eso.

Código 5.11: Añadido botón de reporte a la barra de navegación

```
func updateInterface() {
        //Create report button
        let reportButton = UIBarButtonItem(title: "", style: .plain, target: ↩
            ↪ self, action: #selector(reportActionButton))
        reportButton.image = UIImage(systemName: "exclamationmark.triangle")
        self.navigationItem.rightBarButtonItems = [reportButton]
}

@objc func reportActionButton() {
    CustomReportAlertViewController.shared.showReportAlertViewController(view: ↩
        ↪ self)
}
```

Para la funcionalidad se ha optado por no solo abrir el gestor de correo electrónico de iOS, sino por añadirle la posibilidad de que si el dispositivo no cuenta con esa aplicación, prueba

entre otra lista de posibles opciones.

Código 5.12: Funciones para comprobar el gestor de correo electrónico a utilizar y para enviar el correo

```
func showReportAlertViewController(view: UIViewController) {
          let recipientEmail = "uaplant.app@gmail.com"
          let nameController = NSStringFromClass(view.classForCoder)
          let subject = "Reportar error en " + nameController
          let body = ""

   if MFMailComposeViewController.canSendMail() {
              let mail = MFMailComposeViewController()
              mail.mailComposeDelegate = self
              mail.setToRecipients([recipientEmail])
              mail.setSubject(subject)
          mail.setMessageBody(body, isHTML: false)

              view.present(mail, animated: true)

          } else if let emailUrl = createEmailUrl(to: recipientEmail, subject:↩
              ↪  subject, body: body) {
              UIApplication.shared.open(emailUrl)
          }
}

func createEmailUrl(to: String, subject: String, body: String) -> URL? {
          let subjectEncoded = subject.addingPercentEncoding(↩
              ↪ withAllowedCharacters: .urlHostAllowed)!
          let bodyEncoded = body.addingPercentEncoding(withAllowedCharacters: ↩
              ↪ .urlHostAllowed)!

          let gmailUrl = URL(string: "googlegmail://co?to=\(to)&subject=\(↩
              ↪ subjectEncoded)&body=\(bodyEncoded)")
          let outlookUrl = URL(string: "ms-outlook://compose?to=\(to)&subject↩
              ↪ =\(subjectEncoded)")
          let yahooMail = URL(string: "ymail://mail/compose?to=\(to)&subject↩
              ↪ =\(subjectEncoded)&body=\(bodyEncoded)")
          let sparkUrl = URL(string: "readdle-spark://compose?recipient=\(to)&↩
              ↪ subject=\(subjectEncoded)&body=\(bodyEncoded)")
          let defaultUrl = URL(string: "mailto:\(to)?subject=\(subjectEncoded)↩
              ↪ &body=\(bodyEncoded)")

          if let gmailUrl = gmailUrl, UIApplication.shared.canOpenURL(gmailUrl↩
              ↪ ) {
              return gmailUrl
          } else if let outlookUrl = outlookUrl, UIApplication.shared.↩
              ↪ canOpenURL(outlookUrl) {
              return outlookUrl
          } else if let yahooMail = yahooMail, UIApplication.shared.canOpenURL↩
              ↪ (yahooMail) {
              return yahooMail
          } else if let sparkUrl = sparkUrl, UIApplication.shared.canOpenURL(↩
```

```
                ↪ sparkUrl) {
38              return sparkUrl
39          }
40
41          return defaultUrl
42  }
43
44  func mailComposeController(_ controller: MFMailComposeViewController, ↩
        ↪ didFinishWith result: MFMailComposeResult, error: Error?) {
45          controller.dismiss(animated: true, completion: nil)
46  }
```

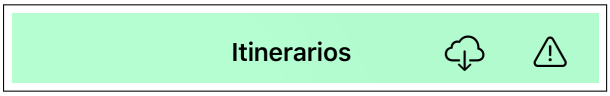

Figura 5.26: Barra de navegación final con el botón de reporte.

Una vez finalizado todo esto, ya estaría la aplicación para iOS completa y totalmente funcional.

5.4 Android

5.4.1 Visión general

En esta sección se va a concretar como se ha procedido al desarrollo de la aplicación de Android de forma nativa.

Además se han utilizado clases auxiliares para la realización de diferentes tareas:

- **Room:** Almacenamiento de datos en el dispositivo de forma persistente.
- **JSON:** Formato del fichero de lectura de los itinerarios del servidor, para lo que se ha implementado un lector especifico para estos datos.
- **Dependencies:** Lista de dependencias en Android Studio, donde podemos agregar frameworks de terceros a nuestro proyecto. Se ha usado para descargar: PlayServicesMaps, PlayServicesLocation, Room, EventBus (GreenRobot (s.f.)) y Picasso (Inc. (s.f.)).

5.4.2 Creación del proyecto

En primer lugar el desarrollo de la aplicación para Android pasa por la descarga y configuración del entorno de desarrollo *Android Studio*, donde vamos a realizar la programación para este sistema operativo de forma nativa al completo.

Posteriormente a la descarga de Android, creamos el proyecto con un nombre clave, en este caso al ser el desarrollo para el TFM se le pondrá como nombre *"TFM-Botanica-Android"*, ajustando la API mínima en 21 (Android 5.0 Lollipop).

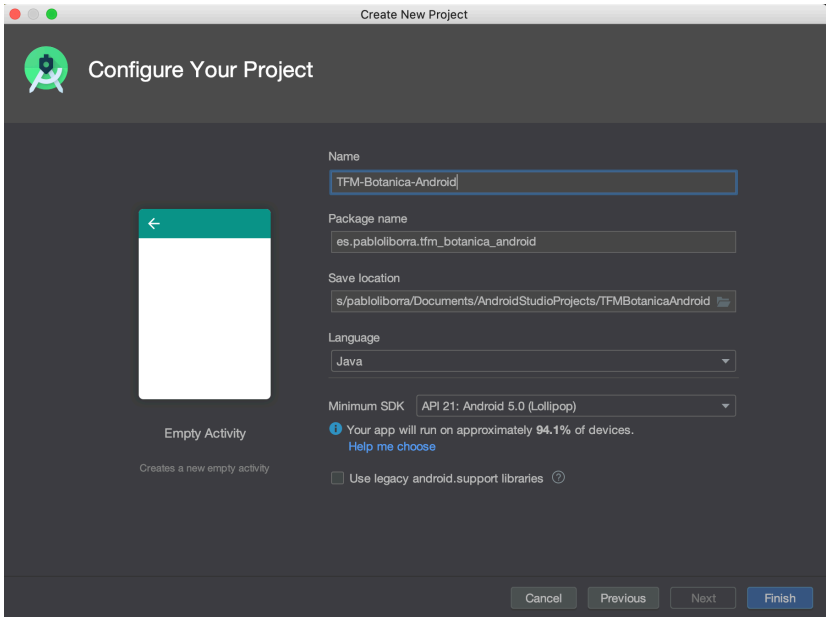

Figura 5.27: Creación del proyecto Android.

Para poder comenzar a crear partes de la aplicación, deberemos crear en Room las entidades necesarias para almacenar los datos. A su vez estas entidades tendrán atributos que las conformarán, como podemos ver en la Figura 5.10, aunque a diferencia de XCode, las entidades y atributos en Android Studio tendrán que conformarse de una manera diferente, pudiendo desaparecer alguna entidad respecto a la aplicación de XCode. Las entidades que crearemos en Room corresponderán a cada tabla necesaria, donde se almacenarán todos los datos, como son:

- Itinerario -> Route
- Actividad -> Activity
- Planta -> Plant
- Pregunta -> Question

Dado que el proceso de desarrollo para Android es posterior al desarrollo en iOS, podemos realizar el desarrollo iterando lo menos posible sobre el modelo inicial, como podemos observar en el capítulo 4, basándonos en el desarrollo para iOS.

5.4.3 Pantalla inicial

Se añade una primera pantalla de bienvenida al usuario, la cual posteriormente servirá para la descarga de itinerarios o rutas en segundo plano.

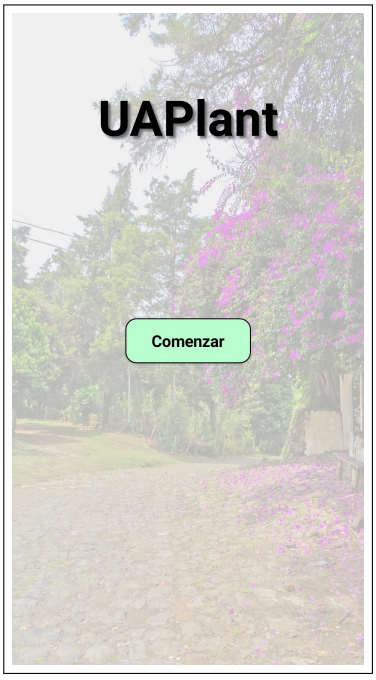

Figura 5.28: Pantalla inicial de la aplicación en Android.

Una vez ya ha cargado la aplicación, sabemos que deseamos tener tres secciones en la pantalla principal, por lo que añadimos un "Tab Bar Activity" con tres pestañas, como podemos ver en la figura 5.29.

Las pestañas añadidas son las siguientes:

- **Itinerarios:** Lista de los itinerarios.
- **Plantas:** Lista de plantas, correspondientes a las actividades de los itinerarios.
- **Créditos:** Contiene información relevante sobre el desarrollo de la aplicación.

Figura 5.29: Barra de navegación con las tres pestañas principales.

Para cada una de estas pestañas del "Tab Bar Activity" creamos un "Fragment" que corresponderá a la vista que se mostrará en el contenido de la actividad según la pestaña seleccionada.

5.4.4 Pantalla de lista de itinerarios

Como se ha dicho anteriormente, para mostrar la lista de itinerarios haremos uso de un "Fragment", el cual se mostrará en el "ViewPager" de la actividad principal.

En nuestro "Fragment" de la lista de itinerarios añadimos un "RecyclerView", el cual nos permitirá crear una lista de "CardViews" con la información que deseemos mostrar, en nuestro caso mostraremos los siguientes datos sobre el itinerario:

- Título
- Descripción breve
- Número de actividades completadas/número de actividades totales

Para poder mostrar el "CardView" con la información deseada, creamos un nuevo layout.xml con el diseño que queramos. Una vez ya tenemos este diseño, como queremos que los itinerarios estén divididos en tres secciones diferentes, necesitamos un nuevo layouts.xml para el diseño de las secciones, por lo que hasta este momento tenemos los siguientes layouts.xml:

- Diseño del CardView con la información del itinerario.
- Diseño de la sección, junto con un RecyclerView donde mostrar los CardView.
- Diseño del Fragment, donde tendremos un RecyclerView para mostrar cada sección con sus CardViews.

De este modo para poder mostrar toda la información deseada, creamos en primer lugar un nuevo archivo java para almacenar toda la información de los CardViews, el cual llamaremos ”RouteListItem”.

Código 5.13: Datos almacenados en la clase RouteListItem para mostrar en el CardView de la lista de itinerarios

```
private Route route; //Ruta asociada al item
private String title; //Titulo de la ruta
private String description; //Descripcion de la ruta
private int completeActivities; //Actividades completadas en la ruta
private int totalActivities; //Total de actividades de la ruta
```

Ahora debemos hacer lo mismo con la sección, por lo que creamos un nuevo archivo java para almacenar los datos que se mostrarán en las secciones.

Código 5.14: Datos almacenados en la clase RoutesSection para mostrar en la sección de la lista de itinerarios

```
private int sectionImage; //Imagen de la seccion
private String sectionTitle;  //Titulo de la seccion
private List<RouteListItem> routesList; //Lista de los ítems (CardViews) que ↩
    ↪ tiene contiene seccion
```

Para mostrar todas nuestras secciones e ítems de la lista de itinerarios debemos crear dos ”Adapter” con sus respectivos ”ViewHolder”, los cuales se encargarán de gestionar los datos en cascada, de manera que el primer Adapter que creamos va a gestionar los ítems de la lista de itinerarios, y el segundo va a gestionar las secciones, donde mostrará a su vez el anterior Adapter con los ítems.

Creamos nuestro ”RouteChildAdapterList”, el cual contendrá a su vez un ”ViewHolder” para gestionar los diferentes ítems con la información de los itinerarios. En el constructor del Adapter le pasamos la lista de ítems, de tipo ”RouteListItem”, que va a contener esa sección junto con un objeto de tipo ”onRouteClickListener”, el cual es una interfaz que creamos para detectar el ”onClick” que realizaremos sobre cada ítem. Para crear nuestros ítems, vamos a irnos a la función ”onBindViewHolder”, donde configuramos cada holder con la información de cada ítem.

Código 5.15: Clase RouteChildAdapterList para la gestión los ítems de las secciones de la lista de itinerarios

```
public class RoutesChildAdapterList extends RecyclerView.Adapter<↩
    ↪ RoutesChildAdapterList.RecyclerRouteHolder> {
   private Context context;
   List<RouteListItem> routesList;
   OnRouteClickListener onRouteClickListener;

   public RoutesChildAdapterList(Context context, List<RouteListItem> ↩
       ↪ routesList, OnRouteClickListener onRouteClickListener) {
       this.context = context;
       this.routesList = routesList;
       this.onRouteClickListener = onRouteClickListener;
   }
   @NonNull
   @Override
   public RoutesChildAdapterList.RecyclerRouteHolder onCreateViewHolder(↩
```

```
        ↪ @NonNull ViewGroup parent, int viewType) {
14      View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.↩
            ↪ item_list_view, parent, false);
15      return new RoutesChildAdapterList.RecyclerRouteHolder(view, this.↩
            ↪ onRouteClickListener);
16  }
17  @Override
18  public void onBindViewHolder(@NonNull RecyclerRouteHolder holder, int ↩
        ↪ position) {
19      holder.title.setText(this.routesList.get(position).getTitle());
20      holder.description.setText(this.routesList.get(position).getDescription↩
            ↪ ());
21      holder.numActivities.setText(this.routesList.get(position).↩
            ↪ getCompleteActivities() + "/" + this.routesList.get(position).↩
            ↪ getTotalActivities());
22  }
23  @Override
24  public int getItemCount() {
25      return this.routesList.size();
26  }
27  class RecyclerRouteHolder extends RecyclerView.ViewHolder implements View.↩
        ↪ OnClickListener {
28
29      TextView title;
30      TextView description;
31      TextView numActivities;
32
33      OnRouteClickListener onRouteClickListener;
34
35      public RecyclerRouteHolder(@NonNull View itemView, OnRouteClickListener ↩
            ↪ onRouteClickListener) {
36          super(itemView);
37
38          this.title = itemView.findViewById(R.id.titleRoutes);
39          this.description = itemView.findViewById(R.id.descriptionRoutes);
40          this.numActivities = itemView.findViewById(R.id.numActivities);
41          this.onRouteClickListener = onRouteClickListener;
42
43          itemView.setOnClickListener(this);
44      }
45      @Override
46      public void onClick(View v) {
47          this.onRouteClickListener.onRouteClick(v, getAdapterPosition());
48      }
49  }
50  public interface OnRouteClickListener {
51      void onRouteClick(View v, int position);
52  }
53}
```

Ahora que ya se ha creado el Adapter que gestiona los ítems, creamos uno nuevo "RouteAdapterList" para las secciones, el cual se gestionará de la misma manera que el anterior, exceptuando que añadimos un "onRouteClick" y que cada "ViewHolder" de esta clase contendrá además una instancia de nuestro "RouteChildAdapterList", lo cual permitirá que cada sección pueda mostrar los ítems añadidos a ella.

Código 5.16: Clase RouteAdapterList para la gestión de las secciones de la lista de itinerarios

```
@Override
    public void onRouteClick(View v, int position) {
        Intent intent = new Intent(this.activity, RoutesMap.class);
        intent.putExtra(Constants.routeExtraTitle, this.routes.get(position).↩
            ↪ getRoute());
        this.activity.startActivity(intent);
    }

    public class RecyclerRouteHolder extends RecyclerView.ViewHolder {
        private ImageView imageSection;
        private TextView titleSection;
        private TextView numActivitiesSection;
        RecyclerView childRecyclerView;
```

Ahora que ya se han creado los Adapter necesarios para mostrar los ítems con sus correspondoentes secciones, solo nos falta ir a nuestro "ListRoutesFragment" y un crear un "RouteAdapterList" con las tres secciones que contiene nuestra lista y el total de ítems que tenemos. Una vez creado nuestro Adapter con los datos necesarios, se lo asignamos al RecyclerView del Fragment de la lista de itinerarios. Podemos observar un ejemplo en la figura 5.30.

La estructura final de los archivos creados es:

- **Clases**
  - ListRoutesFragment
  - RouteListItem
  - RoutesSection
  - RoutesChildAdapterList
  - RoutesAdapterList
- **Layouts**
  - fragment_list_routes
  - section_route
  - item_list_view

5.4.5 Pantalla de lista de plantas

Para crear la lista de plantas realizaremos los mismos pasos que para crear la lista de itinerarios, por lo que creamos sus respectivos layouts.xml de los diseños del Cardview de los ítems y de las secciones. Una vez tenemos ya los layouts de los dieños, creamos los respectivos

Adapter y ViewHolders para los ítems y las secciones, quedando la estructura de la lista de plantas de la siguiente manera. Podemos observar un ejemplo en la figura 5.31.

- **Clases**
  - ListPlantsFragment
  - PlantListItem
  - PlantsSection
  - PlantsChildAdapterList
  - PlantsAdapterList
- **Layouts**
  - fragment_list_plants
  - section_plant
  - item_list_plant_view

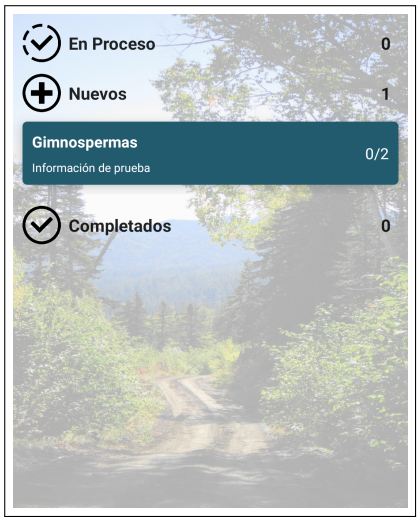

Figura 5.30: Pantalla de la lista de itinerarios.

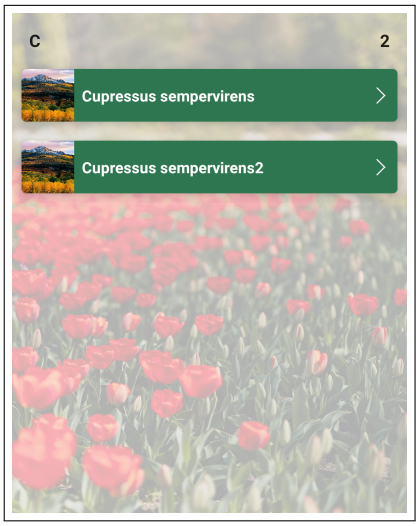

Figura 5.31: Pantalla de la lista de plantas.

5.4.6 Mapa con actividades

Ahora vamos a continuar añadiendo el mapa y los marcadores de las actividades. Para el mapa vamos a utilizar Google Maps para mantener la concordancia con la aplicación de iOS, por lo que para añadir este mapa crearemos una nueva actividad de tipo "Google Maps Activity", lo cual ya nos creará el Fragment y el layout necesarios.

Ahora que ya tenemos el mapa, simplemente crearemos los markers de los activities asociados a la ruta seleccionada, asignando cada marcador según el estado que tenga, pudiendo ser de cuatro diferentes: En proceso, disponible, completada e inactiva, diferenciando cada uno de ellos con un color diferente, y almacenaremos cada marcador en un "LinkedHashMap" con su actividad.

Código 5.17: LinkedHashMap para almacenar los marcadores y las actividades correspondientes del mapa

```
LinkedHashMap<Marker,Activity> markers = new LinkedHashMap<Marker,Activity>();
```

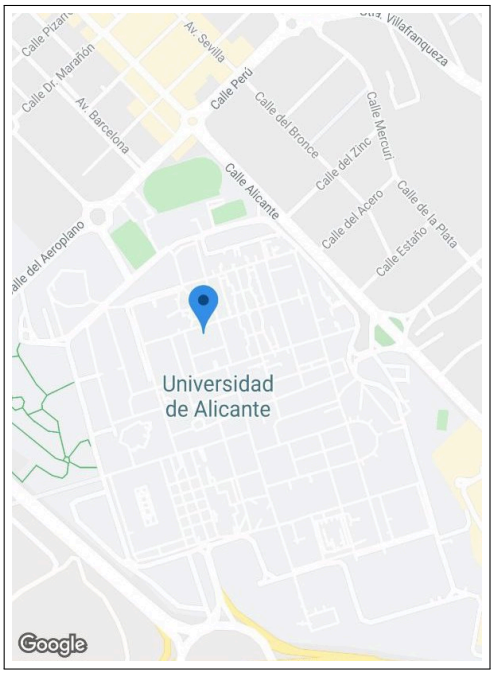

Figura 5.32: Mapa de Google Maps con una actividad.

5.4.7 Ventana con significado del marcador

Una vez añadidos los marcadores de cada actividad correspondiente, creamos la ventana de información que vamos a mostrar cuando pulsemos sobre un marcador. Para crear la ventana, creamos un nuevo layout.xml con el diseño que queremos que tenga.

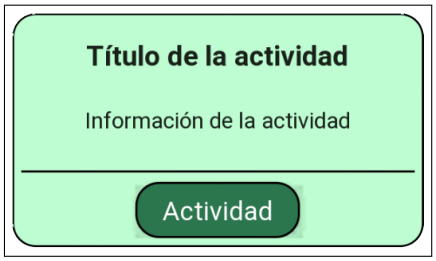

Figura 5.33: Ventana de información del marcador seleccionado.

5.4.8 Ventana informativa con significado de los marcadores

Para aclararle al usuario el significado de los colores de los marcadores y los estados, hemos añadido una pantalla informativa a la que se accederá mediante un botón a la derecha en la barra de navegación.

Figura 5.34: Botón de información de los marcadores en la barra de navegación.

Figura 5.35: Pantalla informativa de los marcadores y los estados de las actividades.

5.4.9 Pantalla con la información de la actividad

Una vez añadido el mapa, creamos una nueva actividad para mostrar la información en detalle de la actividad pulsada. En esta nueva actividad vamos a mostrar los siguientes datos:

- Título de la actividad
- Fotografía de la localización
- Descripción de la actividad
- Color del estado de la actividad
- Estado de la actividad
- Botones
  - Comenzar actividad
  - Preguntas del test

Al darle al botón de comenzar cambiaremos el estado de la actividad, que pasará de ”Disponible” a ”En Proceso”, guardando este cambio automáticamente en Room, de manera que si la actividad que acabamos de comenzar es la primera del itinerario elegido, el propio itinerario también cambiará de estado, de ”Disponible” a ”En Proceso”, actualizando a su vez el marcador del mapa. Podemos ver el flujo del cambio de estado en la figura 5.37.

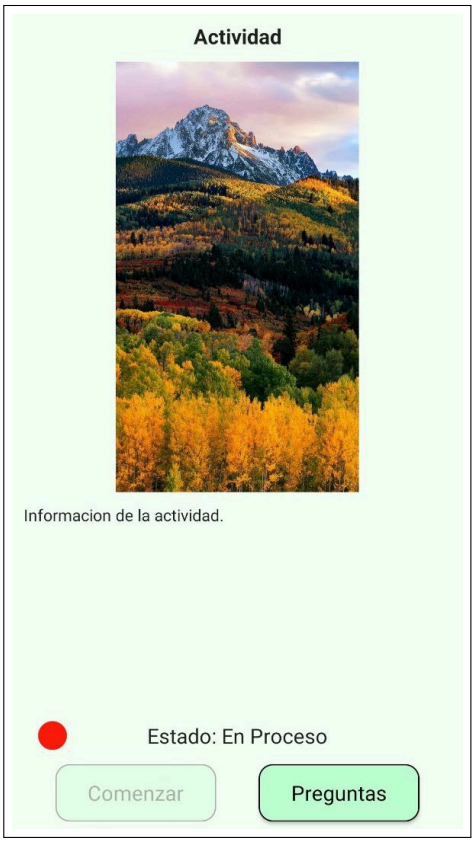

Figura 5.36: Pestaña de información sobre la actividad pulsada.

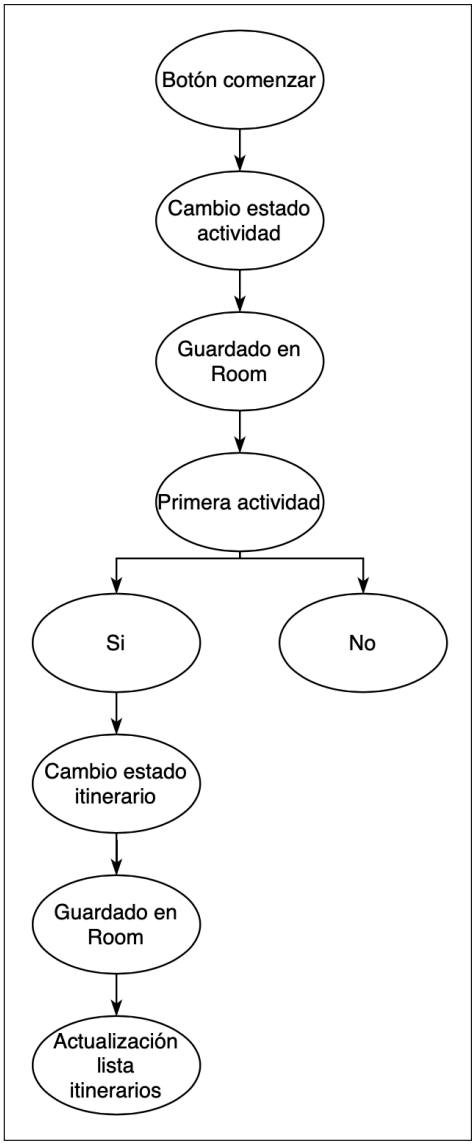

Figura 5.37: Diagrama de flujo de cambio de estado de la actividad.

Una vez ya se ha realizado el cambio de estado y se han comprobado el resto de estados, como se muestra en el diagrama de flujo, procedemos a bloquear el botón de comenzar actividad y a desbloquear el botón que permite realizar el test.

5.4.10 Pantalla con la lista de preguntas y respuestas del test

Para poder realizar el test, creamos una nueva Actividad la cual contendrá un ScrollView, y de dentro de este un RecyclerView y un botón, el cual estará al final del scroll, permitiendo de esta manera que sea un formulario lineal, con las preguntas y al final el botón. Para poder crear la lista de preguntas, realizamos el mismo procedimiento que para crear la lista de itinerarios y de plantas, creando un nuevo ítem con el diseño que queremos, y un nuevo Adapter con su ViewHolder correspondiente. Una vez mostrada la lista de preguntas, configuramos el botón de ”Enviar” que hemos añadido, para que compruebe si todas las respuestas son correctas o no, y abrirá un panel correspondiente a cada caso.

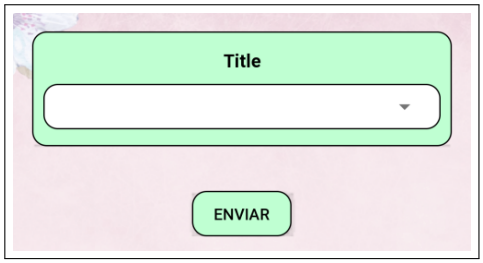

Figura 5.38: Diseño preguntas, mostrando una preguntas y el botón de enviar al final de la lista.

5.4.11 Ventanas de información sobre el resultado del test

Una vez ya tenemos la pantalla de con la lista de cuestiones, habría que realizar la configuración para la corrección de estas preguntas y respuestas, de manera que una vez el usuario pulse el botón de ”Enviar”, cambiaremos el color de fondo del Spinner de las respuestas, de manera que le proporcione al usuario un feedback de que respuestas son correctas y cuáles no. Para que el usuario comprenda el código de colores elegido para las respuestas correctas y las incorrectas, en caso de tener alguna respuesta fallida, se le mostrará un mensaje al usuario indicándole que ha pasado y lo que debe corregir.

Figura 5.39: Alerta indicativa de que el usuario ha fallado alguna respuesta.

Como podemos observar en la figura 5.39, indicamos al usuario que las respuestas marcadas en rojo son incorrectas, y que debe corregirlas para poder finalizar el test. Llegados a ese punto, si el usuario finaliza correctamente el test, se le mostrará otra alerta indicativa diciéndole que ha logrado desbloquear la planta correspondiente, y que tiene la opción de continuar completando actividades o de ir a ver la información de la planta.

Figura 5.40: Alerta indicativa de que el usuario ha finalizado el test correctamente.

Una vez la planta se ha desbloqueado, cambiaremos en Room el atributo ”unlock” de la entidad ”Plant”. Este atributo fue añadido para obligar al usuario a realizar cada actividad correspondiente si este deseaba poder consultar la planta en la lista de estas, por lo que una vez que se ha completado el test, realizamos lo comentado, guardamos este cambio en Room con el objetivo de que el usuario ya disponga de esta planta para futuras consultas.

5.4.12 Pantalla con la información de la planta

Ahora que ya se ha completado el flujo de los itinerarios, las actividades, las preguntas y el desbloqueo de la planta, procedemos a realizar la pantalla de información de la planta, de manera que si la desbloqueamos y pulsamos ”Ver”, como podemos observar en la figura 5.40, o si seleccionamos la planta deseada de la lista de estas debe mostrarnos la información que podemos observar en la figura 5.41.

- Nombre de la planta
- Carrusel de imágenes
- Familia a la que pertenece
- Información relevante

Cabe destacar que en el caso de que el usuario desee información sobre una planta bloqueada, se le mostrará una alerta indicativa de que actividad debe completar para desbloquearla, como la que podemos ver en la figura 5.42.

5.4.13 Pantalla de Créditos

Configuramos los créditos mediante una nueva Actividad, la cual mediante TextViews mostrará información relevante del desarrollo de la aplicación.

- Desarrollador
- Colaboraciones
- Propuesta
- Recursos externos

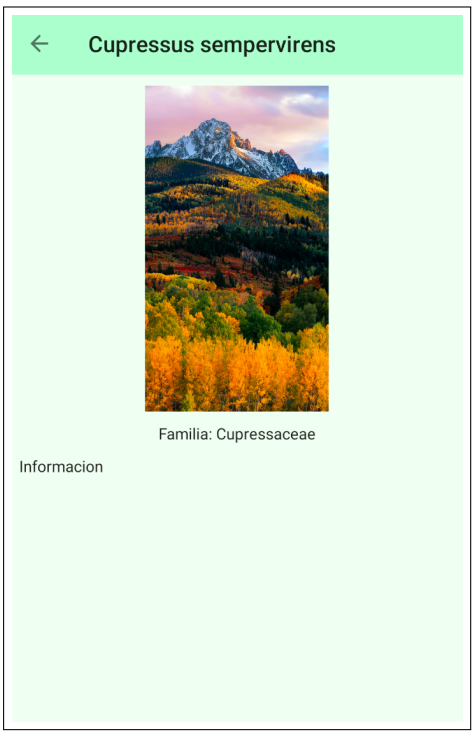

Figura 5.41: Pantalla de información sobre una planta.

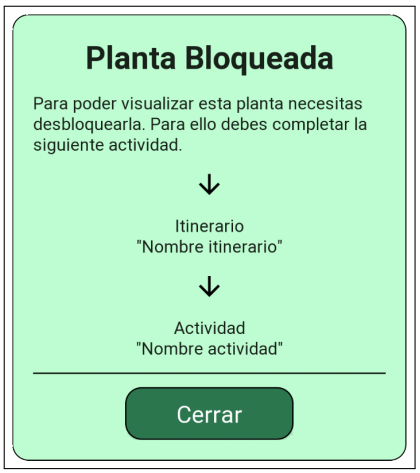

Figura 5.42: Alerta indicativa de que la planta seleccionada esta bloqueada y que actividad debe completar para desbloquearla.

Figura 5.43: Pantalla de créditos con información relevante sobre el desarrollo de la aplicación.

5.4.14 Diseño e iconos de la aplicación

Como el desarrollo de Android se está realizando de forma posterior al desarrollo en iOS, desde un principio se ha procurado mantener el mismo diseño, la misma gama de colores, los mismos fondos y los mismos iconos en ambas aplicaciones, manteniendo así una uniformidad entre aplicaciones como podemos ver entre las figuras 5.29, 5.44 y 5.45, y en las figuras 5.19, 5.20 y 5.21 del desarrollo para iOS.

Figura 5.44: Diseño de las secciones de la lista de itinerarios.

Figura 5.45: Diseño de las secciones de la lista de plantas.

5.4.15 Conexión y lectura de datos del servidor

Como ya se ha explicado en la sección 5.2 de "Servidor y fichero JSON" y en el desarrollo de iOS, realizaremos la conexión al servidor y parsearemos el JSON con la información de los itinerarios. Para realizar estas peticiones en Java utilizamos el comando:

Código 5.18: Ejemplo de petición de datos al servidor mediante HttpURLConnection

```
1 URL url = new URL(urlServer + "index.txt");
2 HttpURLConnection httpURLConnection =(HttpURLConnection) url.openConnection();
3 InputStream inputStream = httpURLConnection.getInputStream();
4 BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(↩
    ↪ inputStream));
```

Para que esta petición funcione correctamente, tenemos que pedir permisos de "Internet" y "Access Network State" en el "AndroidManifest.xml", y ademas en este mismo fichero deberemos incluir "android:usesCleartextTraffic="true"" para que las peticiones "http" funcionen correctamente.

Cuando ya hemos realizado la petición de descarga del ”index.txt”, lo leeremos en busca del nombre de acceso a cada fichero .json disponible, de manera que una vez realicemos esto comprobaremos que ese itinerario no existe previamente en Room, de existir previamente en la memoria no se volvería a guardar, pero si este no existe debemos parsearlo para obtener los diferentes datos del itinerario, de manera que conforme se vayan obteniendo, se irán creando las entidades necesarias y guardandolas en la BBDD. Un ejemplo de como se guardaría en Room la información de la entidad ”Route”, conforme a la estructura creada previamente, sería:

Código 5.19: Ejemplo de guardado de los datos de la entidad ”Route”

```
1 Route route = new Route(JO.get("itinerario").toString(),JO.get("
    ↪ informacion_itinerario").toString(),State.AVAILABLE); //Estado inicial
    ↪ siempre "Available", ya que todas las rutas estarán disponible una vez
    ↪ guardadas.
2 AppDatabase.getDatabaseMain(contextActivity).daoApp().insertAllRoutes(route);
```

5.4.16 Gestor de descarga

Continuando con los datos extraídos del .json, una vez se llegue al apartado de las imágenes, crearemos un gestor de descargas, igual que se hizo en iOS, el cual comenzará la descarga de las imágenes de forma inmediatamente posterior al guardado de todos los datos necesarios de las diferentes entidades asociadas al itinerario. Para que este gestor pueda realizar las descargas de las diferentes imágenes, mientras se vayan guardando todos los datos iremos guardando en dos diccionarios diferentes la información necesaria para ello. Por un lado tendremos un primer diccionario que almacenará la url de la imagen de la localización de la actividad, y la propia actividad donde guardarla, y por otro lado tendremos otro diccionario que almacenará las urls de las diferentes imágenes del carrusel de la planta, y la propia planta donde almacenar esas imágenes.

Código 5.20: Diccionarios LinkedHashMap donde se almacenan las urls de las imágenes a descargar en el gestor

```
1 LinkedHashMap<Activity,String> imagesLocalizationToDownload = new
    ↪ LinkedHashMap<Activity,String>();
2 LinkedHashMap<Plant,List<String>> imagesPlantsToDownload = new LinkedHashMap<
    ↪ Plant,List<String>>();
```

Para realizar las peticiones de descarga de las urls de las imágenes anteriormente guardadas, utilizaremos la librería ”Picasso”, la cual mediante una simple función nos permitirá descargar la imagen. Una vez descargada cada imagen, crearemos una carpeta para el itinerario descargado y las almacenaremos en el ”Internal Storage”, permitiendo tener así una estructura organizada para cada itinerario. Una vez que ya se han descargado y guardado todas las imágenes necesarias desde el gestor de descargas, actualizaremos las actividades y las plantas con la ruta de almacenaje de sus imágenes, permitiendo de esta manera poderlas localizar en la memoria para su posterior carga.

5.4.17 Pantalla y banner de descarga de archivos

Ahora que ya se han configurado las diferentes peticiones al servidor, procedemos a añadir una pantalla de descarga que avisará al usuario de que se están descargando los diferentes itinerarios del servidor. Para mostrarle al usuario correctamente esta pantalla de descarga, en cada momento habrá que diferenciar los dos posibles lugares desde los que se podrán realizar estas descargas:

- **Pantalla inicial de bienvenida:** Se mostrará a pantalla completa un diálogo que especificará los archivos descargados/totales a descargar. Podemos ver un ejemplo de esta pantalla en la figura 5.46.
- **Pantalla de la lista de itinerarios:** Aparecerá un banner con un diálogo que especificará los archivos descargados/totales a descargar. Podemos observar un ejemplo este banner en la figura 5.47.

Figura 5.46: Pantalla de descarga de archivos en la pantalla de bienvenida de la aplicación.

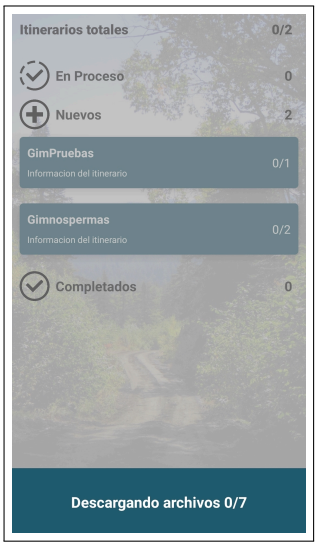

Figura 5.47: Banner de descarga de archivos en la lista de itinerarios.

Una vez ya se han añadido los informativos de las descargas que se están realizando, como en el anterior desarrollo para iOS, solo nos faltaría por añadir:

- Botón para realizar la descarga de archivos desde la lista de itinerarios.
- Borrado de un itinerario.
- Botón para avisar de un posible error en la aplicación.
- Enviar aviso de error por correo.

Añadimos el botón para realizar la descarga de archivos desde la lista de itinerario, de manera que la barra de navegación tendría el siguiente aspecto:

Figura 5.48: Barra de navegación de la lista de itinerarios con el botón para realizar las descargas del servidor.

5.4.18 Borrado de un itinerario

Para borrar un itinerario disponible, añadimos en nuestro ”RouteChildAdapterList” de la lista de rutas, en el ”onBindViewHolder”, que si realizamos una pulsación larga sobre el itemView del holder abra una nueva ventana de información sobre el borrado, y en caso de seleccionar ”Eliminar”, realizamos una Query para borrar los item necesarios de Room.

Código 5.21: OnLongClickListener para el borrado de un itinerario en onBindViewHolder del RouteChildAdapterList

```
holder.itemView.setOnLongClickListener(new View.OnLongClickListener() {
        @Override
        public boolean onLongClick(View v) {
                showDeleteAlert(routesList.get(position).getRoute(), position);
                return false;
        }
});
```

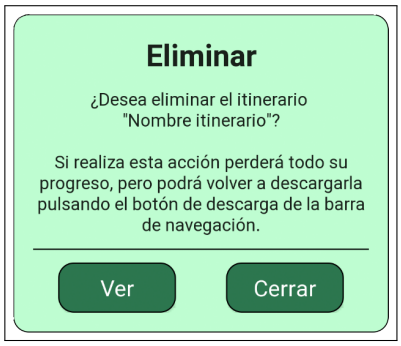

Figura 5.49: Alerta de borrado de un itinerario

Una vez se ha seleccionado que se desea borrar la ruta y todos sus items, borramos ademas del ”Internal Storage” la carpeta que se creó en la descarga de la ruta con todos sus archivos, de manera que liberaremos la memoria. Posteriormente a la eliminación de la ruta y sus datos, es necesaria la comunicación con el ”Fragment” de la lista de plantas para que esta actualice sus Adapter y elimine los ítems de las plantas eliminadas con la ruta, por lo que para lograr esta comunicación se ha utilizado la dependencia ”EventBus”, la cual permite una rápida y eficaz comunicación entre diferentes elementos de la aplicación mediante subscripciones a los eventos.

5.4.19 Envío de una notificación de error mediante correo electrónico

Por último, para finalizar la aplicación, faltaría añadir el botón para avisar de un error y la funcionalidad de este mismo enviando un correo. El botón para abrir el gestor de correo electrónico, lo debemos añadir en todas las pestañas de nuestra aplicación, dándole la posibilidad al usuario de reportar un posible error en cualquier punto de esta. A este botón a su vez deberemos añadirle la funcionalidad de que abra el gestor de correo electrónico, para lo que se ha realizado un "Intent" que capture todos los gestores de correo instalados en el sistema.

Código 5.22: Gestor de reporte por correo electrónico

```
1 private void sendEmailReport() {
2       Intent i = new Intent(Intent.ACTION_SENDTO);
3       i.setType("message/rfc822");
4       i.setData(Uri.parse("mailto:?subject=" + "Reportar error en " + '"' + ↩
            ↪ ACTIVITY_TAG + '"' + "&to=" + "uaplant.app@gmail.com"));
5
6       startActivity(Intent.createChooser(i, "Enviar
7 }
```

Figura 5.50: Barra de navegación final con el botón de reporte.

Una vez realizado todo esto, ya estaría la aplicación para Android completa y totalmente funcional.

6 Resultados

Aquí se pueden observar capturas de los resultados finales de la aplicación, de manera que la imagen de la izquierda corresponde a capturas en un iPhone 11 Pro Max con el sistema operativo iOS 13.3, y las capturas de la derecha a un Xiaomi MI9 con el sistema operativo Android 10 con MIUI 12. Ademas podemos encontrar el código fuente la aplicación para ambos sistemas operativos en: Iborra (s.f.-b) y Iborra (s.f.-a).

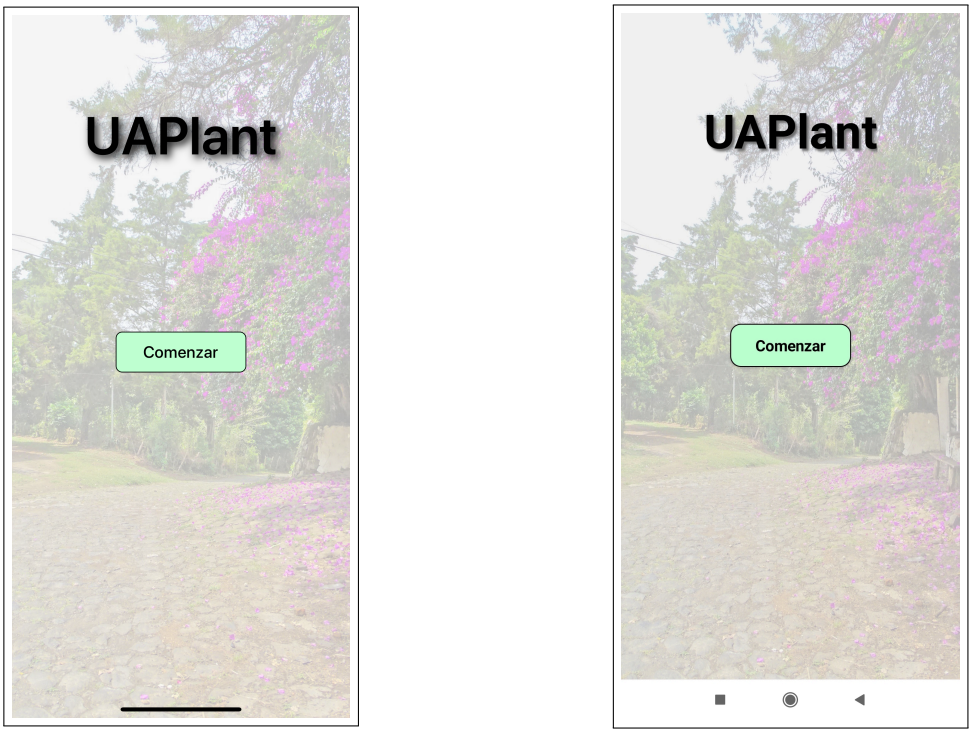

Figura 6.1: Pantalla inicial.

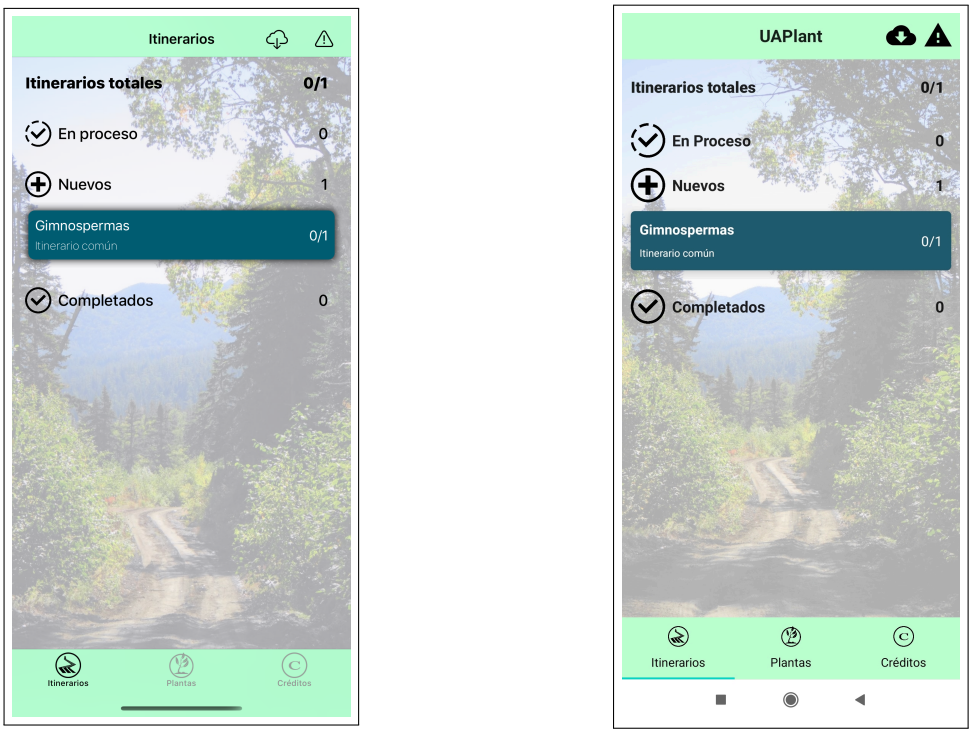

Figura 6.2: Lista de itinerarios con un itinerario.

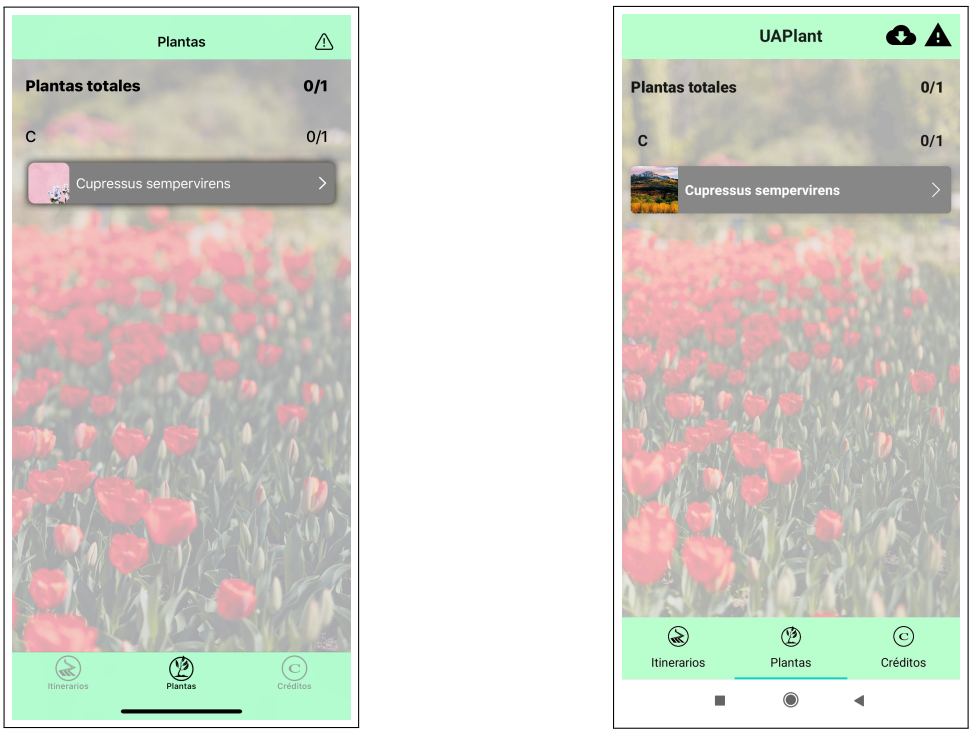

Figura 6.3: Lista de plantas con una planta bloqueada.

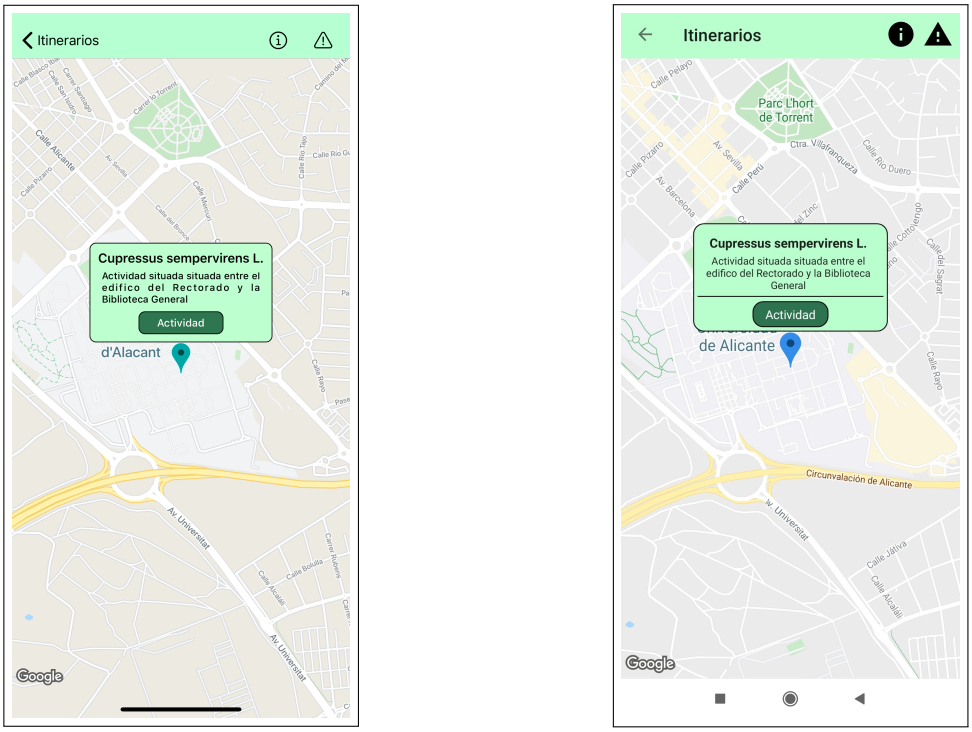

Figura 6.4: Mapa con una actividad y su información.

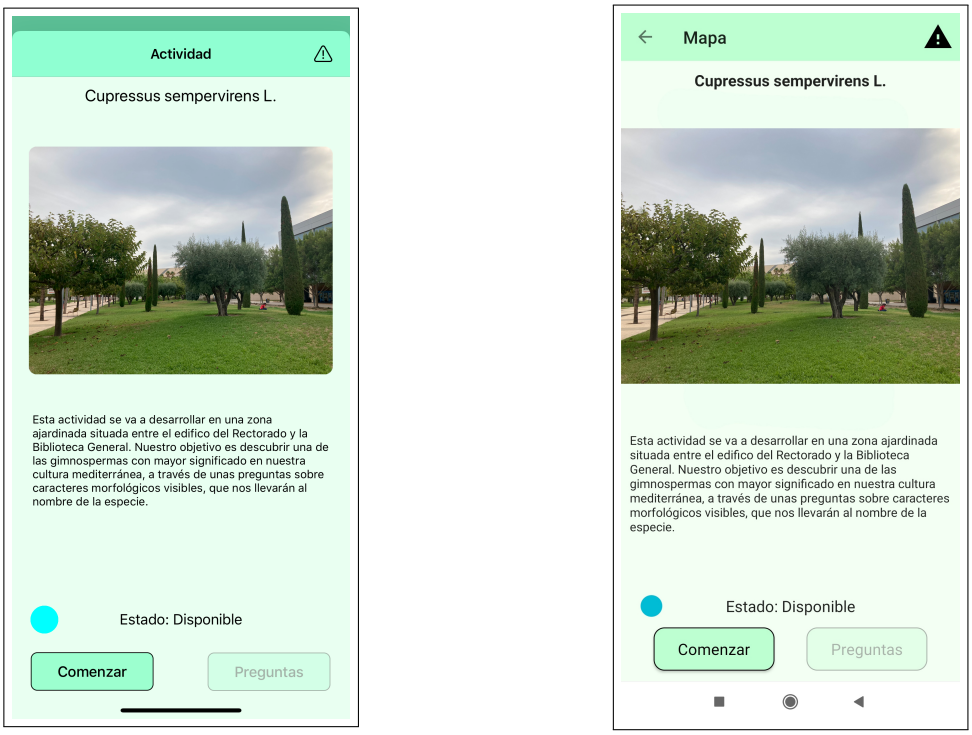

Figura 6.5: Pantalla con los detalles de la actividad elegida.

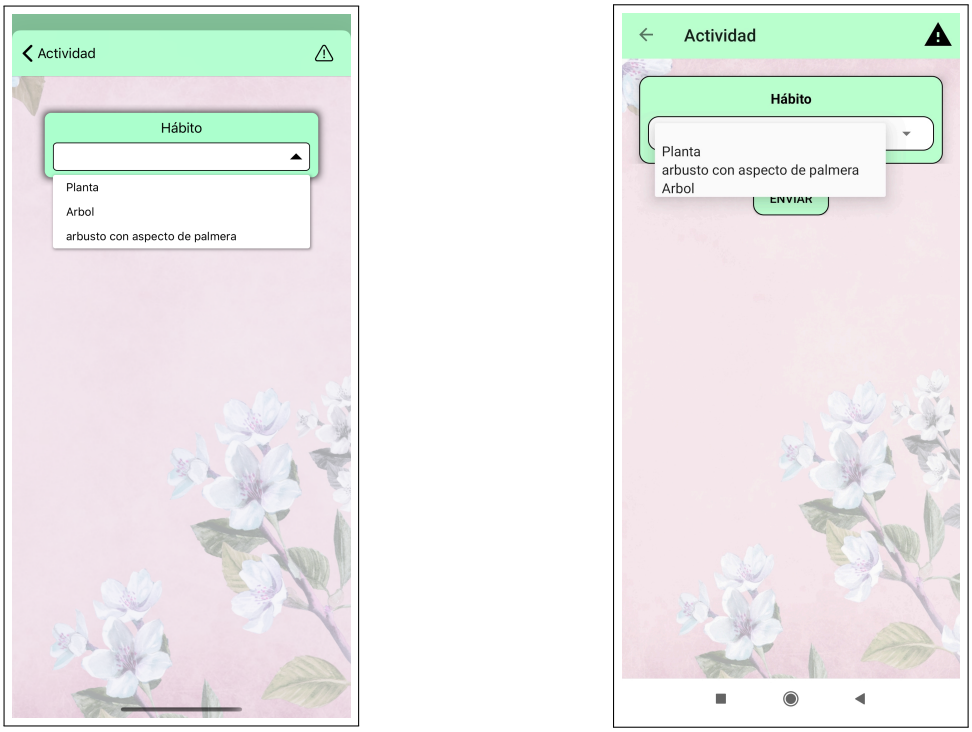

Figura 6.6: Test con la lista de preguntas de la actividad.

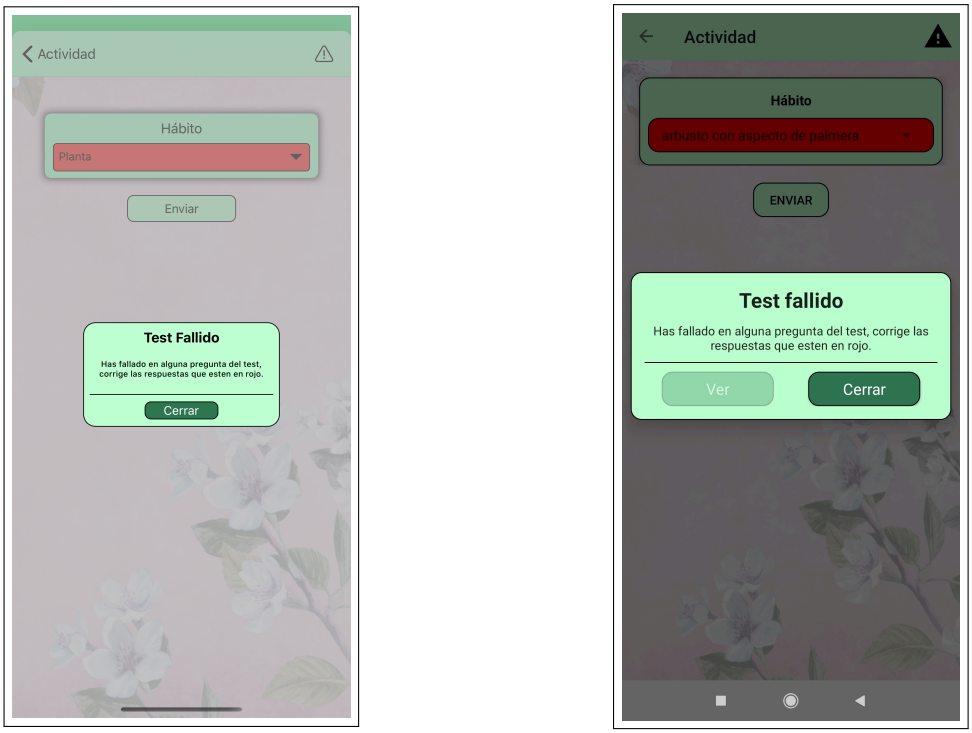

Figura 6.7: Ventana informativa de que se ha fallado alguna respuesta en el test.

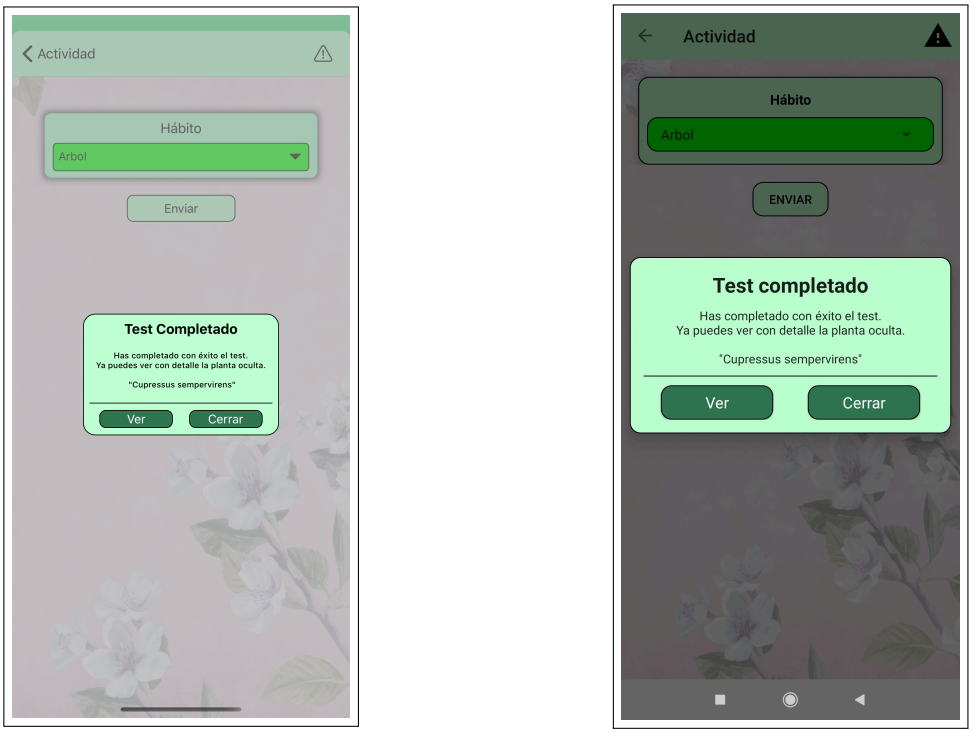

Figura 6.8: Ventana informativa de que se ha completado el test correctamente.

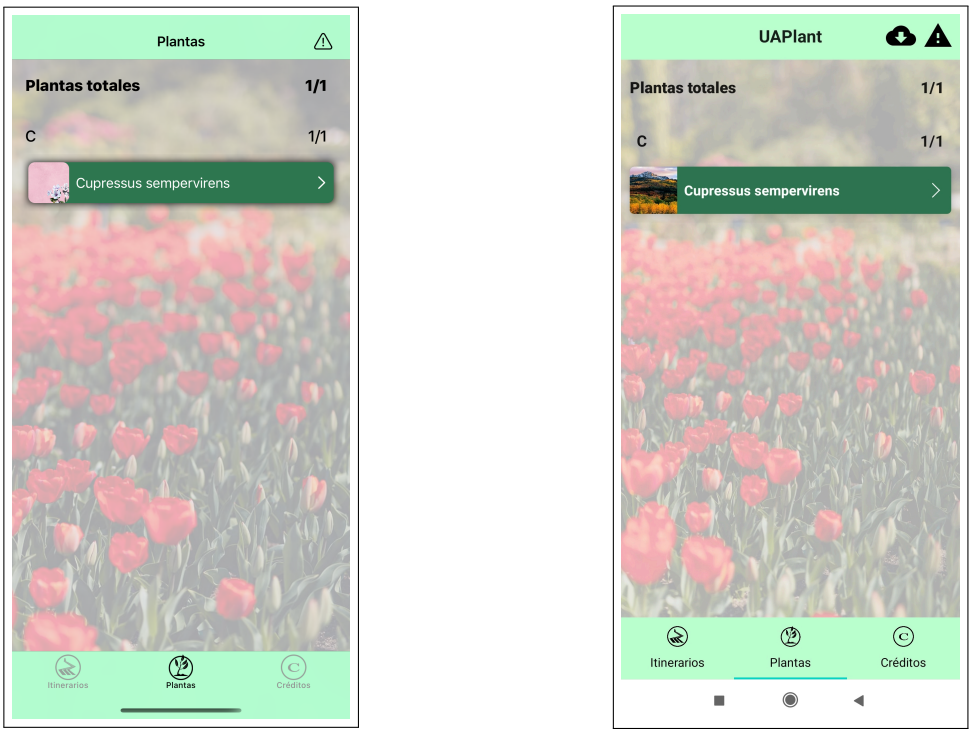

Figura 6.9: Lista de plantas con una planta desbloqueada.

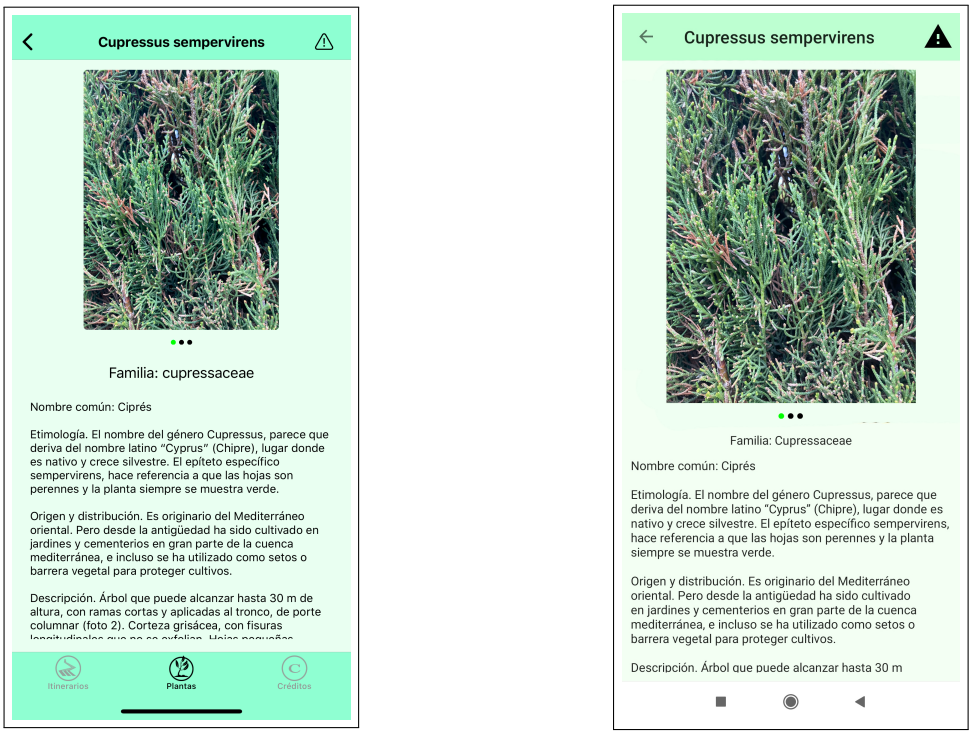

Figura 6.10: Pantalla con la información de la planta seleccionada.

Figura 6.11: Pantalla de créditos con información sobre el desarrollo de la aplicación.

7 Conclusiones

En este capítulo se va a explicar el motivo final por el que se han desarrollado dos versiones nativas de la aplicación, las diferencias principales que se han encontrado en el desarrollo para iOS y Android, detallando las principales dificultades, y además una lista con las futuras mejoras. Comentar que primero se realizó el desarrollo para iOS y posteriormente para Android, permitiendo así poder realizar una comparativa de dificultad.

En un primer lugar destacar que se decidió realizar dos versiones nativas, tanto para iOS como para Android, y no una versión híbrida, dado que estaba previsto que la aplicación incluyese el reconocimiento de imágenes, por lo que el desarrollo nativo permitía una mejor inclusión de la red neuronal en cada sistema operativo, y por otro lado se buscaba que tuviera una alta disponibilidad, y dado que la aplicación esta orientada a ser utilizada en el campus de Universidad de Alicante (UA), una versión de desarrollo híbrida no era viable dado que la cobertura wifi no funciona correctamente en todo el perímetro del campus, así pues guardar la información en local parecía una mejor opción. Además la API de mapas nativa permite una mejora navegación por el campus, utilizando así la versión de Google Maps para cada sistema operativo.

Por otro lado destacar que la mayor dificultad encontrada en el desarrollo de iOS ha sido con la descarga de datos del servidor, en especial las imágenes, ya que hasta que se creó un servidor propio y se configuró todo correctamente la conexión de datos que se realizaba no funcionaba correctamente, por lo que no descargaba nada, este proceso lo podemos observar entre las secciones 5.2 y 5.3.15. Además comentar también la dificultad con algunas partes más concretas como son el dropdown de las preguntas y el carrusel de las imágenes de la pantalla de detalle de la planta, para los que se debió acabar usando un pod externo.

En el caso de Android, la primera dificultad que se encontró fue creando la base de datos con Room, ya que el entendimiento de como funciona y la necesidad de un alto conocimiento de uso de esta librería, dificultaron en gran medida la creación de las entidades y las relaciones. Por otro lado también se encontró una dificultad creando las listas principales, ya que el entendimiento de como funcionaban correctamente los Adapter, y el tener que utilizar dos de ellos en cascada para poder realizar las secciones y los ítems, fue bastante complejo y dificultoso. Destacando las dificultades citadas, el desarrollo de la aplicación para ambos sistemas operativos ha sido satisfactorio, ya que se han aprendido muchos conocimientos nuevos en la creación de aplicaciones completas para ambos sistemas operativos.

Por último comentar que esta aplicación continuará con su futuro desarrollo, ya que al estar en colaboración directa con el Departamento de Ciencias Ambientales y Recursos Naturales (DCARN), se desea que esta sea lo mas completa posible, por lo que se podrían destacar como futuros añadidos:

- Optimización de la lectura y guardado de datos de la BBDD.
- Mejora y optimización de la lectura de datos del servidor.
- Reconocimiento de imágenes mediante una red neuronal.
- Utilización de la geolocalización del usuario, asegurando así que los itinerarios deban ser realizados en el lugar correspondiente del mapa.
- Más información en los detalles de las actividades y las plantas mediante nuevos controles de navegación.
- Añadido de mas alertas visuales para una mayor interacción usuario-aplicación.
- Posibilidad de cambiar la forma de la vista de las listas.
- Añadido de una BBDD de usuarios registrados y no registrados, permitiendo a los usuarios registrados realizar nuevas funcionalidad:
  - Guardar itinerarios favoritos.
  - Compartir itinerarios con otros usuarios registrados.
- Distinción de usuarios registrados entre alumnos, profesores o visitantes.
- Panel de control para usuarios profesores, permitiendo añadir itinerarios desde la propia aplicación.

Bibliografía

Anubha Pearline, S., Sathiesh Kumar, V., y Harini, S. (2019). A study on plant recognition using conventional image processing and deep learning approaches. *Journal of Intelligent & Fuzzy Systems*, *36*(3), 1997–2004.

Arquitectura ios. (s.f.). Descargado de `https://sites.google.com/site/tecnologiaiostm/desarrollo-de-aplicaciones/arquitectura-ios`

Clave dicotómica. (s.f.). Descargado de `https://es.wikipedia.org/wiki/Clave_dicotómica`

Duran, W. (s.f.). *Las mejores app móviles para identificar plantas.* Descargado de `https://wwwhatsnew.com/2019/10/12/las-mejores-app-moviles-para-identificar-plantas/`

Google. (s.f.). *What is android?* Descargado de `https://www.android.com/intl/es_es/what-is-android/`

GreenRobot. (s.f.). *Eventbus.* Descargado de `https://greenrobot.org/eventbus/`

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., … Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Iborra, P. L. (s.f.-a). *Uaplant android.* Descargado de `https://github.com/PabloLIborra/TFM-Botanica-Android`

Iborra, P. L. (s.f.-b). *Uaplant ios.* Descargado de `https://github.com/PabloLIborra/TFM-Botanica-iOS`

Inc., S. (s.f.). *Picasso.* Descargado de `https://square.github.io/picasso/`

<jriosdev@gmail.com>, J. R. (s.f.). *iosdropdown.* Descargado de `https://github.com/jriosdev/iOSDropDown`

Pearl Doughty-White, M. Q. (s.f.). *Codebases, million of lines of code.* Descargado de `https://www.informationisbeautiful.net/visualizations/million-lines-of-code/`

shima11. (s.f.). *Flexiblepagecontrol.* Descargado de `https://github.com/shima11/FlexiblePageControl`

Lista de Acrónimos y Abreviaturas

BBDD	Base de Datos.
DCARN	Departamento de Ciencias Ambientales y Recursos Naturales.
DCCIA	Departamento de Ciencia de la Computación e Inteligencia Artificial.
IA	Inteligencia Artificial.
TFM	Trabajo Final de Máster.
UA	Universidad de Alicante.
WWDC	Apple Worldwide Developers Conference.